



FAULT AND DEFECT TOLERANT COMPUTER ARCHITECTURES:
RELIABLE COMPUTING WITH UNRELIABLE DEVICES

DISSERTATION

George R. Roelke IV, Major, USAF

AFIT/DS/ENG/06-07

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

The views expressed in this document are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the United States Government.

AFIT/DS/ENG/06-07

FAULT AND DEFECT TOLERANT COMPUTER ARCHITECTURES:
RELIABLE COMPUTING WITH UNRELIABLE DEVICES

DISSERTATION

Presented to the Faculty
Graduate School of Engineering and Management
Air Force Institute of Technology
Air University
Air Education and Training Command
In Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy

George R. Roelke IV, B.Cmp.E., M.S.C.E.

Major, USAF

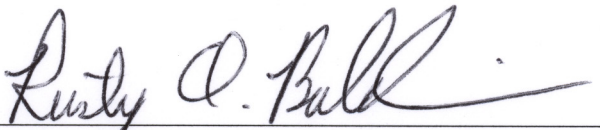
September 2006

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

FAULT AND DEFECT TOLERANT
COMPUTER ARCHITECTURES:
RELIABLE COMPUTING
WITH UNRELIABLE DEVICES

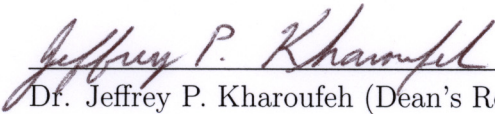
George R. Roelke IV, B.Cmp.E., M.S.C.E.
Major, USAF

Approved:



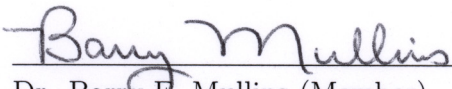
Dr. Rusty O. Baldwin (Chairman)

15 Aug 06
date



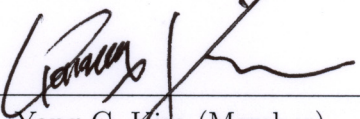
Dr. Jeffrey P. Kharoufeh (Dean's Representative)

17 Aug 06
date



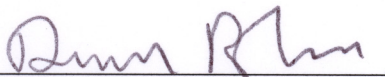
Dr. Barry E. Mullins (Member)

16 Aug 06
date



Dr. Yong C. Kim (Member)

15 Aug 06
date



Dr. Dursun A. Bulutoglu (Member)

15 Aug 06
date

Accepted:

M. U. Thomas
Dean, Graduate School of Engineering and Management

Date

Abstract

As conventional silicon Complementary Metal-Oxide-Semiconductor (CMOS) technology continues to shrink, logic circuits are increasingly subject to errors induced by electrical noise and cosmic radiation. In addition, the smaller devices are more likely to degrade and fail in operation. In the long term, new device technologies such as quantum cellular automata and molecular crossbars may replace silicon CMOS, but they have significant reliability problems. Rather than requiring the circuit to be defect-free, fault tolerance techniques incorporated into an architecture allow continued system operation in the presence of faulty components.

This research addresses construction of a reliable computer from unreliable device technologies. A system architecture is developed for a “fault and defect tolerant” (FDT) computer. Trade-offs between different techniques are studied, and the yield of the system is modelled. Yield and hardware cost models are developed for the fault tolerance techniques used in the architecture.

Fault and defect tolerant designs are created for the processor, and its most critical component, the cache memory. A content-addressable memory (CAM)-based cache design is developed. Simulation results show the cache achieves 90% yield with device failure probabilities of 3×10^{-6} , three orders of magnitude better than non fault tolerant caches of the same size. The entire processor achieves 70% yield with device failure probabilities exceeding 10^{-6} . The hardware redundancy required to achieve this performance is approximately 15 times that of a non-fault tolerant design. While large compared to fault tolerant designs used today, this architecture allows the use of devices much more likely to fail than silicon CMOS. Given the size improvements predicted for future device technologies, the hardware overhead may be acceptable.

As part of the work to develop reliable models for fault tolerance techniques, an improved model is developed for NAND Multiplexing, a cornerstone fault-tolerance

technique based upon large levels of redundancy. The model is the first exact model for NAND Multiplexing with small and medium amounts of redundancy. Previous models are extended to account for dependence between the inputs and produce more accurate results. An example shows the required hardware redundancy is reduced by 50%.

Acknowledgements

I would like to thank my advisor, Dr. Rusty Baldwin. He was a guiding force and sanity check throughout this long process. From scoping the initial research problem to reviewing journal articles with his mighty red pen, he was instrumental in successful completion of my research. I would also like to thank the members of my committee: Dr. Barry Mullins, Dr. Yong Kim, and Dr. Dursun Bulutoglu. Their helpful comments and guidance made for better papers, more complete results, and a happier PhD student.

Finally, I would like to thank my lovely wife. Words fail me, but without her steadfast support and love, none of this would have been possible.

George R. Roelke IV

Table of Contents

	Page
Abstract	iv
Acknowledgements	vi
List of Figures	xiv
List of Tables	xx
List of Symbols	xxii
List of Abbreviations	xxiii
 I. Introduction	 1
1.1 Background	1
1.1.1 The Problem of Yield	2
1.1.2 The Problem of Errors	3
1.2 Renewed Interest in Fault Tolerant Computing	4
1.3 Statement of Purpose	5
1.3.1 Research Focus	5
1.3.2 Applications	5
1.4 Defect Tolerant Computing	6
1.4.1 Potential Solutions	6
1.4.2 A Notional Architecture	8
1.5 Summary	10
 II. Background	 11
2.1 Overview	11
2.2 Challenges in Nanometer Scale CMOS	11
2.2.1 The Silicon Roadmap	12
2.2.2 Circuit Effects	15
2.2.3 Potential CMOS Solutions	22
2.2.4 Limitations of the Current Design Paradigm	37
2.3 New Device Technologies	39
2.3.1 Next Generation Devices	40
2.3.2 Common Features of Developmental Devices	50
2.4 Fault Tolerance	51
2.4.1 Faults and Testing Concepts	51
2.4.2 Fault Tolerance Strategies	57

	Page
2.4.3 Hardware Techniques for Fault Tolerance	60
2.5 Radiation Effects	72
2.5.1 Causes	73
2.5.2 Effects	75
2.5.3 Relation to Process Scaling	78
2.5.4 Solutions	79
2.6 Fault Tolerant Architectures	80
2.6.1 Fault and Defect Tolerant Systems	80
2.6.2 Fault Tolerant FPGAs	84
2.6.3 FPGA Testing	95
III. Research Goals	105
3.1 Motivation	105
3.1.1 Four Goals	105
3.2 Goal 1: Develop the FDT System Architecture	107
3.2.1 Questions Addressed	107
3.2.2 Quantifiable Goals	108
3.3 Goal 2: Design the Functional Architecture	109
3.3.1 Questions Addressed	109
3.3.2 Quantifiable Goals	110
3.4 Goal 3: Map the Functional Architecture onto Emerging Technologies	112
3.4.1 Questions Addressed	112
3.4.2 Quantifiable Goals	113
3.5 Goal 4: Develop an accurate analytical model for NAND Multiplexing	114
3.5.1 Questions Addressed	115
3.6 Research Contributions	115
3.7 Summary	116
IV. Methodology	117
4.1 Problem Scope	117
4.2 Goal 1: Develop the FDT System Architecture	118
4.2.1 Tasks	118
4.2.2 Scope and Parameters	119
4.3 Goal 2: Design the Functional Architecture	120
4.3.1 Tasks	120
4.3.2 Evaluation	121
4.4 Goal 3: Map the Functional Architecture to Emerging Technologies	121
4.4.1 Tasks	121

	Page
4.4.2 Evaluation	122
4.5 Goal 4: Develop an analytical model for NAND Multiplexing	122
V. Tools and Models	123
5.1 Yield Models	123
5.1.1 Basic Yield Models	123
5.1.2 Clustering	123
5.1.3 Multiple Components	126
5.2 Fault Models	127
5.3 Key Fault Tolerance Techniques	128
5.3.1 NAND Multiplexing	128
5.3.2 R-Modular Redundancy	129
5.3.3 Modular Reconfiguration	130
5.3.4 TMR-Protected Reconfiguration	131
5.3.5 Threshold Gate Logic	132
5.4 Memory Array Fault Tolerance	132
5.4.1 Error Correcting Codes	132
5.4.2 Global Spares/Content Addressable Memories	135
5.4.3 Spare Rows and Columns	135
5.5 Hardware Cost Models	136
5.5.1 Primitives	137
5.5.2 Fault Tolerance Circuits	139
5.6 Summary	140
VI. von Neumann Multiplexing	141
6.1 Introduction	141
6.2 von Neumann Multiplexing	142
6.2.1 Overview	142
6.2.2 Han and Jonker Analytical Model	145
6.2.3 Sadek, Nikolic, and Forshaw's Analytical Expression	147
6.3 Dependency Between the Outputs of the NAND Array	148
6.4 Updated Distribution Model	153
6.4.1 Derivation of S_Z	153
6.4.2 Derivation of the distribution of S_Q	155
6.5 New Fault Types: Input and Output Stuck-At Faults	158
6.5.1 Output Stuck-At Faults	158
6.5.2 Input Stuck-At-Zero	160
6.5.3 Input Stuck-At-One	160

	Page
6.6 Simulation	161
6.6.1 Simulation Setup	161
6.6.2 Simulation Results	164
6.7 An Application	173
6.8 Conclusion	174
VII. High Level Architectures	176
7.1 Introduction	176
7.2 Desired Characteristics	176
7.3 Fault Tolerance Strategies	178
7.3.1 Module/Circuit Level	178
7.3.2 Instruction Level	181
7.3.3 Operating System Level	183
7.3.4 Application Level	185
7.4 Proposed Architecture	186
7.5 Concept of Operation	189
7.5.1 Yield Testing	189
7.5.2 Startup Configuration	190
7.5.3 Runtime Testing	191
7.5.4 Soft Error Recovery	192
7.5.5 The Role of the Operating System	192
7.6 Conclusion	193
VIII. Cache Memory	194
8.1 Introduction	194
8.2 Background	195
8.2.1 Cache Characteristics	195
8.2.2 Yield Modelling	195
8.2.3 Hardware Fault Tolerance Techniques	195
8.2.4 Previous Fault Tolerant Memories	197
8.2.5 Error Correcting Codes	198
8.2.6 Support Logic Fault Tolerance	198
8.2.7 Assumptions	199
8.3 Architectures for Comparison	199
8.3.1 Cache A: Non Fault Tolerant Fixed Design	199
8.3.2 Cache B: Basic CAM Design	201
8.4 FDT Cache Memory Overview	212
8.4.1 CAM Top Level Design	214
8.4.2 CAM Word Design	215
8.4.3 Hardware Cost	217

	Page
8.5	Operation and Testing 217
8.5.1	Normal Operation 217
8.5.2	Cache-Specific Error Behaviors 218
8.5.3	Testing and Recovery 218
8.6	Design Choices 222
8.6.1	Error Correcting Codes 222
8.6.2	Bus Design 223
8.6.3	ECC effect on CAM Array and Output Bus Yield 229
8.6.4	BIST/R and Control Modules 231
8.6.5	Other Modules 232
8.7	Yield Simulation 233
8.7.1	Simulation Methodology 233
8.7.2	Cache C Results 234
8.8	Conclusions 236
8.8.1	Comparison to Other Caches 236
8.8.2	Hardware Cost Comparison 236
8.8.3	SEU Performance and Operational Reliability 238
8.8.4	The Big Picture 239
IX.	Core Logic 240
9.1	Introduction 240
9.2	Methodology 240
9.2.1	General Approach 240
9.2.2	Assumptions 241
9.3	FDT CPU Design 241
9.3.1	Initial Architecture 241
9.3.2	Analytical Model 245
9.4	Yield Performance 246
9.4.1	Simulation Description 246
9.4.2	Simulation Results 249
9.5	Conclusions 255
X.	Device Mapping 256
10.1	Introduction 256
10.2	Technology Choices 257
10.3	QCA Background 258
10.3.1	QCA Basics 258
10.3.2	QCA Logic Design 262
10.3.3	QCA Defects 265
10.3.4	QCA Fault Tolerant Architectures 268

	Page
10.4 Area, Power, and Speed Estimation	268
10.4.1 Differences from CMOS	268
10.4.2 Area Estimation	269
10.4.3 Power Estimation	273
10.4.4 Speed Estimation	273
10.5 A Defect Model for QCA	275
10.6 Fault Tolerant Circuits for QCA	277
10.6.1 TMR	277
10.6.2 Reconfiguration	281
10.6.3 TMR-Protected Reconfiguration	284
10.7 Performance Comparison	287
10.7.1 Setup	287
10.7.2 Observations	288
10.7.3 Designing for Fault Tolerance	304
10.8 Conclusions	305
XI. Conclusions and Recommendations	307
11.1 Conclusions	307
11.1.1 Goal Status	307
11.1.2 The Big Picture	308
11.2 Contributions	309
11.3 Recommendations for Future Research	310
Appendix A. Additional Background	313
A.1 Programmable Logic Devices	313
A.1.1 Families of Programmable Logic Devices	317
A.1.2 Field Programmable Gate Arrays	320
A.1.3 Dynamic and Partial Reconfiguration	334
A.1.4 Placement and Routing	338
A.2 Reconfigurable Computing	347
A.2.1 Applications	349
A.2.2 Typical Architectures	349
A.2.3 Application Development	352
A.2.4 Limitations	356
Appendix B. Long Term Group Research Goals	358
B.1 Four Research Phases	358
B.1.1 Current Research Focus and Shortcomings	361
B.2 Phase 2: Improve Fault Tolerance in Current FPGAs	362
B.2.1 Applicable Background	362

	Page
B.2.2 Questions Addressed	364
B.2.3 General Goals	365
B.2.4 Research Contributions	365
B.2.5 Demonstration	366
B.3 Phase 3: Dynamic Reconfiguration	367
B.3.1 Applicable Background	367
B.3.2 Questions Addressed	368
B.3.3 General Goals	368
B.3.4 Research Contributions	369
B.3.5 Demonstration	369
B.4 Phase 4: Intelligent Agents	370
B.4.1 Applicable Background	370
B.4.2 Questions Addressed	370
B.4.3 General Goals	370
B.4.4 Research Contributions	371
B.4.5 Demonstration	371
B.5 Summary	372
Bibliography	374
Index	391

List of Figures

Figure		Page
1.1.	Emerging Technology Vectors.	7
1.2.	System Architecture Diagram.	9
2.1.	CMOS Node Size Definitions.	13
2.2.	A Typical MOSFET (Cross Section).	13
2.3.	Detailed MOSFET (Cross Section).	16
2.4.	Normalized Delay vs. V_t/V_{dd} Ratio.	19
2.5.	Increasing Effects of Interconnect Delay.	22
2.6.	Sublithographic Patterning.	24
2.7.	80nm Lines Patterned with X-Ray Lithography.	25
2.8.	Halo Doping.	26
2.9.	Retrograde Channel Doping.	27
2.10.	Vertical Transistor.	29
2.11.	Two Gate FET Designs.	30
2.12.	Strained NMOS Silicon MOSFET.	31
2.13.	PMOS and NMOS Strained Silicon Designs.	32
2.14.	Intel's Strained Silicon Approach.	33
2.15.	Damascene Gate Cross Section.	36
2.16.	Damascene Gate Fabrication Process.	37
2.17.	Molecular Crossbar.	42
2.18.	Generic CMOL Circuit. Design.	44
2.19.	Single Electron Transistor Design.	46
2.20.	Carbon Nanotube FET.	47
2.21.	Quantum Cell.	48
2.22.	QCA Majority Gate.	48
2.23.	QCA Memory Cell and Adder Design.	49

Figure		Page
2.24.	RMR Performance.	63
2.25.	NAND Multiplexer	66
2.26.	Combined DWC/CED Scheme.	68
2.27.	Combined DWC/CED Operation.	69
2.28.	Comparison of Hardware FT Methods.	72
2.29.	Rad Hard SRAM Cell Using RC Filtering.	87
2.30.	Asymmetric-0 SRAM Cell.	88
2.31.	DICE Memory Cell (Concept).	89
2.32.	DICE Memory Cell.	90
2.33.	FPTA Node Design.	91
2.34.	Horse and King Shifting.	94
2.35.	Column Shifting for CLB Sparing.	95
2.36.	FPGA Interconnect Testing.	100
2.37.	RSTARS Concept.	102
2.38.	RSTARS Tile Rotations.	103
3.1.	Three Research Goals.	106
5.1.	Modular Reconfiguration.	131
5.2.	TMR-Protected Reconfiguration.	133
6.1.	NAND Multiplexer	144
6.2.	Output PDF for 7th Restorative Stage(M=15) for N=20 [WR vs. WOR models]	164
6.3.	Output PDF for 7th Restorative Stage(M=15) for N=20 [PRISM and Combin. Models]	165
6.4.	Reliability vs Error Probability (N=40)	166
6.5.	Reliability vs Number of Stages (N=40)	166
6.6.	Reliability vs Error Probability (N=20)	167
6.7.	Output PDF Residuals 1st Rest. Stage (N=20)	167
6.8.	Residual Reliability vs. Error Probability (N=20)	168

Figure		Page
6.9.	Residual Reliability vs. Number of Stages (N=20)	168
6.10.	Expected Percentage of Incorrect Outputs (N=20)	169
6.11.	Reliability vs Error Probability (N=20)	170
6.12.	Reliability vs Number of Stages (N=20)	170
6.13.	Reliability vs Number of Stages (oSA1)	171
6.14.	Reliability vs Error Probability (oSA0)	172
6.15.	Reliability vs Number of Stages (oSA0)	172
7.1.	Adoption Timeline.	186
7.2.	Top Level Processor Architecture.	188
8.1.	Single Read Port Cache	196
8.2.	Cache A Yield With No Fault Tolerance (Analytical)	201
8.3.	Cache A Yield With No Fault Tolerance (Simulated)	202
8.4.	CAM Cache Architecture 1	203
8.5.	CAM Word Design 1	204
8.6.	Minimum Required Spare CAM Words for Cache B	207
8.7.	CAM Cache B Yield Depending on the Number of Spare Rows	208
8.8.	CAM Array Only Yield for 1024KB Cache	209
8.9.	Support Logic Yield for Cache B	210
8.10.	Bus TGATE Yield for Cache B	211
8.11.	CAM Cache B Yield with 10% Sparing	213
8.12.	Improved CAM Cache Architecture	214
8.13.	Improved CAM Word	215
8.14.	Inset A: CAM Word Architecture Data Bus Controllers	216
8.15.	Redundant Bus Connections	221
8.16.	ECC code choices	224
8.17.	512 bit Yield for Various ECC Strategies	225
8.18.	Non Fatal Bus Fault	227
8.19.	Probability All Output Busses Are Correctable for Cache C	230

Figure		Page
8.20.	TMR-Protected Reconfiguration in the BIST/R Module	232
8.21.	BIST/R Yield vs Module Size	233
8.22.	Cache C Simulated Yield (1024KB)	234
8.23.	Minimum Number of Spare Rows	235
8.24.	Simulation Yield for Cache C	237
8.25.	Simulation Yield for Cache C (Zoomed)	237
9.1.	Yield vs. Chip Kill Total.	241
9.2.	MIPS 32 bit CPU design	242
9.3.	FDT CPU Yield (confidence intervals)	249
9.4.	FDT Core Yield (no Caches)	250
9.5.	FDT CPU Yields (All cases).	251
9.6.	Cache Yields (All Cases).	251
9.7.	FDT CPU Yield (64KB caches).	252
9.8.	FDT CPU Yield (512KB caches)	253
9.9.	FDT CPU Yield (1MB caches).	253
10.1.	QCA Logic States	259
10.2.	QCA Majority Gate.	260
10.3.	QCA Inverter.	264
10.4.	QCA Wire Crossing.	264
10.5.	QCA OR Array.	266
10.6.	Spice-Like Node Model.	270
10.7.	QCA 3-Module Input Layout	278
10.8.	QCA TMR Output Layout	279
10.9.	QCA R-Module Input Layout	281
10.10.	QCA Reconfiguration Output Layout	282
10.11.	QCA TMR-R Output Layout	285
10.12.	QCA TMR Yield Comparison	288
10.13.	QCA TMR MADP Comparison	289

Figure		Page
10.14.	QCA TMR MADP versus Module Size	290
10.15.	QCA TMR MADP versus Input Width	290
10.16.	QCA Reconfiguration versus W_{in}	291
10.17.	TMR MADP versus W_{in}	292
10.18.	QCA MADP vs W_{in}	293
10.19.	TMR MADP versus Output Width	294
10.20.	QCA TMR MADP versus Output Width	294
10.21.	Reconfiguration MADP versus W_{out}	295
10.22.	QCA MADP vs W_{out}	296
10.23.	TMR MADP versus QCA Cell Density	297
10.24.	TMR Yield versus QCA Cell Density	298
10.25.	QCA TMR MADP versus $k_{silicon}$	299
10.26.	TMR MADP for $k_{silicon}$	300
10.27.	Reconfiguration Yield versus R	301
10.28.	QCA FT Model Comparison	302
10.29.	CMOS vs. QCA Fault Tolerance	303
A.1.	Generic FPGA Structure.	315
A.2.	PLA Structure.	315
A.3.	PROM Structure.	319
A.4.	Generic FPGA Structure.	321
A.5.	FPGA CLB Designs.	321
A.6.	Mapping of a Circuit to a Lookup Table.	322
A.7.	Simple FPGA Configurable Logic Block.	323
A.8.	Detailed FPGA CLB Design (Lookup Table Uses).	323
A.9.	Detailed FPGA CLB Design (Slices and Logic Blocks).	324
A.10.	FPGAs With Embedded Cores.	325
A.11.	Typical FPGAs Routing Resources.	326
A.12.	FPGA Design Process.	326

Figure		Page
A.13.	FPGA Serial Configuration.	328
A.14.	ROCR Routing Algorithm.	346
A.15.	Loosely Coupled Reconfigurable Computer.	350
A.16.	Reconfigurable Coprocessor.	351
A.17.	RC Hybrid Processor Architecture.	352
A.18.	Reconfigurable Processor.	353
A.19.	Stages in a Reconfigurable Compiler.	355
A.20.	Runtime Computation for a Reconfigurable Computer.	357
B.1.	Enabling Technologies for an FDT Computer.	359
B.2.	Solution Strategies.	360
B.3.	Layers of Abstraction.	363

List of Tables

Table		Page
1.1.	Defect and Fault Tolerance Layers	8
2.1.	ITRS Roadmap 2003.	15
2.2.	Constant Field Scaling Rules.	18
2.3.	Properties of Selected High K Dielectrics.	34
2.4.	Emerging Logic Devices.	40
2.5.	Radiation Effects on CMOS	75
2.6.	Stuck-Open/Closed Fault Detection in CLB Muxes Using V_t	92
5.1.	Hardware Cost Primitives	138
6.1.	NAND Truth Table for Relevant Errors	158
6.2.	PDF Computation Times for $N = 40, M = 15$	173
7.1.	Applicability of FT Techniques	179
8.1.	Cache Characteristics	195
8.2.	Cache A (No Fault Tolerance) Components	200
8.3.	Cache A (No Fault Tolerance) Maximum Allowable Defect Probability	201
8.4.	Cache B (Simple CAM) Device Count	206
8.5.	Cache C Device Count	217
8.6.	MADP for Bus Design Options.	229
8.7.	Cache C Module MADPs (Simulated)	236
8.8.	Final MADP Results	238
8.9.	Total Device and Redundancy Requirements	238
9.1.	Non-FT CPU Modules (Part One)	243
9.2.	Non-FT CPU Modules (Part Two)	244
9.3.	FDT CPU Modules (Part One)	247
9.4.	FDT CPU Modules (Part Two)	248

Table		Page
9.5.	Final Simulation Results	254
9.6.	Hardware Cost Comparison	255
10.1.	QCA Hardware Costs	271
10.2.	QCA Defect Probabilities	276
A.1.	Conventional Routing Resource Requirements	338
A.2.	On-Chip Router Resource Requirements	347
B.1.	Long Term Research Phases	373

List of Symbols

Symbol		Page
V_t	Threshold Voltage	16
V_{dd}	Power Supply Voltage	16
Y	Yield	55
p_f	Probability of device failure	61
Q_{CRIT}	Critical Charge	76
λ_1	Probability of failure of a single device (transistor)	125
$k_{silicon}$	Silicon clock area scaling constant	272
θ_{qca}	QCA cell fill fraction	272
W_{out}	Number of wires in the module output	280
W_{in}	Number of wires in the module input	280
γ	Pins per logic cell ratio	341
β	Pins per net ratio	341
L	Average wire length	342

List of Abbreviations

Abbreviation		Page
ASIC	Application Specific Integrated Circuit	12
ASIP	Application Specific Instruction Processor	348
ASRAM	Asymmetric Static Random Access Memory	87
BIOS	Built-In Operating System	7
BIST	Built-In Self Test	55
CED	Concurrent Error Detection	7
CMC	Configuration Memory Cell	330
CMOL	CMOS Nanowire MOlecular hybrid	43
CMOS	Complementary Metal-Oxide-Semiconductor	1
CONOPS	Concept of Operations	105
CTMR	Cascaded Tri-Modular Redundancy	63
DISC	Dynamic Instruction Set Computer	351
DRAM	Dynamic Random Access Memory	12
DSP	Digital Signal Processor	348
DUT	Device Under Test	99
DWC	Duplication With Comparison	7
EEPROM	Electrically Erasable Programmable Read Only Memory	318
FIT	Failure Unit	57
FPGA	Field Programmable Gate Array	5
FPTA	Field Programmable Transistor Array	90
FT	Fault Tolerance/Tolerant	11
GPP	General Purpose Processor	69
HDL	Hardware Description Language	325
I/O	Input-Output	54
ITRS	Internation Technology Roadmap for Semiconductors	12

Abbreviation		Page
JTAG	Joint Test Action Group	54
LUT	Lookup Table	320
MADP	Maximum Allowable Defect Probability	108
MOSFET	Metal-Oxide-Semiconductor Field Effect Transistor .	1
MPU	Microprocessor Unit	12
MRAM	Magnetic Random Access Memory	318
MTBF	Mean-Time-Between-Failures	4
MTTF	Mean-Time-To-Failure	56
MTTR	Mean Time To Repair	56
NCD	Native Circuit Description	332
OS	Operating System	7
PAL	Programmable Array Logic	319
PLA	Programmable Logic Array	319
PLD	Programmable Logic Device	313
QCA	Quantum Cellular Automata	1
RMR	R-Modular Redundancy	61
RSFQ	Rapid Single Flux Quantum	1
RSTAR	Roving Self-Test Areas	101
SEE	Single Electron Event	86
SEFI	Single Event Functional Interrupt	330
SER	Soft Error Rate	52
SET	Single Electron Transistor	1
SEUR	Single Event Upset Rate	331
SEU	Single Event Upset	73
SOI	Silicon On Insulator	28
SRAM	Static Random Access Memory	318
TFT	Thin Film Transistor	28
TMRSO	TMR with Shifted Operands	268

Abbreviation		Page
TMR	Tri-Modular Redundancy	61
TREC	Test and Reconfiguration Controller	101
VHDL	VHSIC Hardware Description Language	325
VHSIC	Very High Speed Integrated Circuit	325
VLSI	Very Large Scale Integrated circuit	2
VPR	Versatile Place and Route	341

FAULT AND DEFECT TOLERANT COMPUTER ARCHITECTURES: RELIABLE COMPUTING WITH UNRELIABLE DEVICES

I. Introduction

1.1 *Background*

In 1965, Gordon Moore observed that the number of transistors on a microchip doubled every 12 months [Moo65]. He predicted this rate of increase would likely continue at least until 1975. As it happened, this prediction has actually held true for forty years, becoming the most famous “law” in microelectronics. And yet, almost from its inception, there are those who have predicted the end of Moore’s Law. They cite ever higher technical barriers and predict progress cannot be maintained beyond the next process generation. However, solutions have always been found to these problems, extending the life of the the silicon Complementary Metal-Oxide-Semiconductor (CMOS) transistor.

In spite of progress, there are fundamental limits on how small a Metal-Oxide-Semiconductor Field Effect Transistor (MOSFET) can be made. The oxide layer that separates the gate from the channel is already only atoms thick. Within the next several process generations, this layer will be only a single atom thick. At this size, controlling the flow of current through the transistor becomes extremely difficult. The use of alternative materials, such as Hafnium Oxide [Bou03] may extend the life of the MOSFET by a generation, but in the end, new device families will be required.

Moore’s Law may not end, but its future will likely lie with devices other than silicon CMOS. Single electron transistors (SET), quantum cellular automata (QCA), rapid single flux quantum logic (RSFQ), and carbon nanotube transistors are all being studied as potential replacements for the silicon MOSFET. Each of these devices has been demonstrated on a small scale. While these devices have the potential to be smaller, faster, and consume less power than silicon CMOS, they bring their own

disadvantages. These devices are difficult to fabricate reliably, more susceptible to electrical noise, more likely to fail in operation, and can even require special cooling (e.g., via liquid nitrogen).

Even with conventional CMOS, it is becoming more difficult to fabricate reliable circuits. Quantum effects are becoming important in device performance and conventional scaling rules used to design Very Large Scale Integrated (VLSI) circuits are nearing their limits. The small numbers of atoms in the channels of modern transistors means that even small variations in the number of dopant atoms will significantly change its operating characteristics. In addition, the lithographic techniques used to create circuit patterns are reaching their limits, making it more difficult to create the necessary distinct features. Thus, even with conventional silicon CMOS, it is becoming more difficult and costly to produce smaller, higher performance circuits.

1.1.1 The Problem of Yield. The fundamental challenge in semiconductor fabrication has always been to produce microchips with ever an increasing number of devices in a reliable and economical manner. The conventional paradigm invests enormous amounts of money and resources into fabrication facilities to create chips with few defects. A wafer of chips is fabricated, cut apart, and tested. Those that do not function correctly are discarded. The percentage of correctly functioning chips is known as the *yield*. Modern silicon fabrication yields typically range from 60-85%.

With the exception of memory chips and other specialized circuits, most circuits are not designed to operate correctly in the presence of defective transistors or interconnect wiring on the chip. Even one defect can render the chip unusable. Thus, the defect rate of individual devices must be kept extremely low, requiring precise control over the fabrication process. Each new generation of fabrication facilities costs more than the last, with modern fabrication lines costing in excess of a billion dollars [DeJ98].

The yield of a process can be simply modelled. If defects occur independently, the probability, $P_{functional}$, of a defect-free chip composed of N transistors, each defective with probability P_{def} , is

$$P_{functional} = (1 - P_{def})^N. \quad (1.1)$$

For a modern microprocessor containing $N = 100 \times 10^6$ transistors, a typical defect rate is $P_{def} = 10^{-9}$. Thus, the probability of a defect free chip is $P_{functional} = 0.905$. Increase in the defect probability to $P_{def} = 10^{-8}$, however, and the yield falls to 36.8%.

As the number of transistors on each chip has increased, the difficulty of testing for correct operation has increased as well. Modern microprocessors contain more than 100 million transistors, so testing each device for correct operation is not feasible. Thus, the field of VLSI testing has developed to find ways to quickly and efficiently verify chips. Each manufacturer strikes a balance between the time and money spent testing chips and the penalty for shipping defective devices to a customer.

1.1.2 The Problem of Errors. In addition to being more difficult to fabricate initially, smaller CMOS devices are more susceptible to failures in operation. For example, electromigration makes chips more likely to permanently fail over time. In addition, smaller devices are more influenced by electrical noise in power supplies and by radiation strikes. Alpha particles, neutrons, heavy ions, and electrons have long been a problem for devices operating in space. When cosmic radiation strikes a device, a small electrical pulse is created that may cause logic errors in the device. While the effects on the device are not permanent, these *soft errors* can alter the results of a computation or cause a program crash. Long-term or high intensity radiation exposure can cause permanent damage.

Although computer architects have largely ignored radiation effects for terrestrial applications, they will have to deal with these problems in the future. Indeed, Mi-

Microsoft recently proposed the use of Error-Correcting Code-protected RAM for general purpose computers running the upcoming Windows Vista operating system [Cou06]. Smaller devices have smaller node capacitances and will be affected by radiation and noise. So while process engineers may solve some problems and continue to provide low defect rates, the problems caused by noise and radiation cannot be easily solved at the foundry.

1.2 Renewed Interest in Fault Tolerant Computing

Given the difficulty and cost required to fabricate reliable devices, it is reasonable to ask whether a reliable microchip can be fabricated from inherently *unreliable* devices. Whether constructed from new technologies such as quantum cellular automata or from advanced silicon CMOS, devices in the future will likely suffer higher fabrication defect rates, permanent failures after fabrication, and temporary soft errors in operation. So rather than requiring all devices to be reliable all of the time, is it possible to construct a circuit that can tolerate these device errors?

As it happens, the problem is not new. Early computers were constructed from vacuum tubes which were prone to failure. In fact, Mean-Time-Between-Failures (MTBF) was often measured in hours. Computer architects overcame these problems by incorporating *redundancy* into the design, including Triple Modular Redundancy (TMR) and multiplexing [vN56]. Essentially, TMR performs an operation three times and uses the “best two out of three” results to produce a correct output.

Fault tolerance and defect tolerance became less important following the advent of the integrated circuit. The need for defect tolerance (i.e., the ability to operate with devices that are defective at fabrication) was reduced since microchips could be fabricated reliably and had very low failure rates in operation. Fault tolerance (i.e., the ability to operate in the presence of soft errors) continued to be important for devices used in space applications as well as in areas where reliability was critical (e.g., nuclear power plants, certain military uses, etc.). But with the increasing cost and difficulty of creating reliable devices, interest in fault and defect tolerance is returning.

More recently, the field of reconfigurable computing has emerged as an enabling technology. Reconfigurable computing typically uses programmable logic devices such as a Field Programmable Gate Array (FPGA). Field programmable logic devices provide a computing fabric that can be modified during operation to implement different circuit functions. Run-time reconfiguration of an FPGA has been used for defect and fault tolerance. If a portion of the microchip becomes defective, the application circuit is reconfigured to use other portions of the FPGA.

1.3 Statement of Purpose

1.3.1 Research Focus. The goal of this research is to develop a computer architecture to compute reliably using device technologies that are less reliable than current silicon CMOS. The research involved several areas: fault tolerant computing, defect tolerant computing, reconfigurable computing, VLSI circuit design and test, and computer architecture. The outcome of this research is largely independent of the underlying device technology, assuming only that the devices are smaller, more difficult to fabricate, and more subject to operational failures than current silicon CMOS.

1.3.2 Applications. This research enables application circuits to function correctly even when implemented on device technologies prone to soft and hard errors. It is an enabling technology in nanoscale CMOS (i.e., CMOS with feature sizes of less than 100 nanometers) and other new device technologies. Without this capability, it will be impossible to economically build reliable circuits with these technologies. Defect tolerant architectures indirectly improve the yield of conventional fabrication processes, and thereby lower manufacturing and test costs. In the past, the validity of Moore's Law relied on progress from device and process engineers; it will soon also require support from the computer architect.

Initially, the first use of this research will be in space and military applications where reliability is critical. FPGAs are heavily used in these areas, and a great deal of

research has been directed to making them more reliable. Aspects of this research are easily implemented with FPGAs. Thus, it is relevant even with today's silicon CMOS devices. As process size decreases, silicon CMOS will be less reliable, making progress in the other areas essential to reliable computing. As discussed in later chapters, it may be impossible to build chips from non-CMOS technologies without incorporating defect and fault tolerance.

1.4 Defect Tolerant Computing

This section briefly examines the problem as well as potential solutions. A reliable computer requires the ability to both detect and correct failures. Section 1.4.1 summarizes the layers of hardware abstraction for research, while Section 1.4.2 presents a notional architecture for a FDT computer. This general model is explained in later chapters and used as the operating paradigm for the FDT computer. Development of this paradigm, or concept of operations, is one of the goals of this research. The goals are listed in detail in Chapter III.

1.4.1 Potential Solutions. The problem of system reliability can be addressed at several levels. There are those who debate the necessity of incorporating fault and defect tolerance at the architectural level, citing the past successes of the device engineers at low defect fabrication. Figure 1.1, shows three areas of long-term technology research: “non-classical” CMOS technologies, alternative device families, system architectures [Bou03]. Reliability can be achieved at the device level, but may become prohibitively expensive. As shown in the figure, defect tolerant computing addresses the problem of reliable computing at the architectural level.

As discussed in Section 1.3.2, defect tolerant computing is a viable research area regardless of the device technology used in the future. While process engineers may be successful at producing circuits with high yield, it is likely to be extremely expensive. Defect tolerant microchips can be fabricated with less rigorous process controls, at much lower cost. The computer architect would be given a tool to make tradeoffs

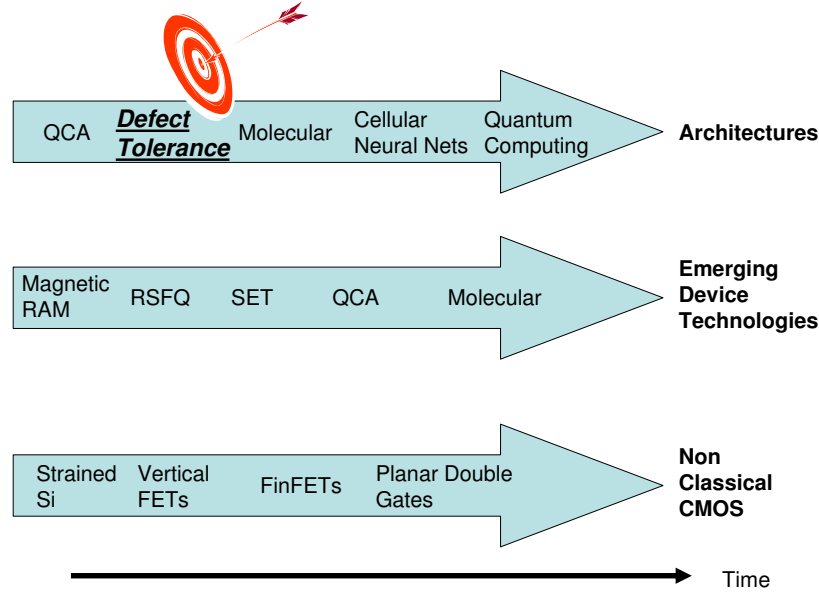


Figure 1.1: Emerging Technology Vectors [HBZB02]. Successors to silicon CMOS are being examined at many levels. The goal of this research is to address fault and defect tolerance at the architectural level.

between the costs (e.g, performance, power, area, economic) of redundant hardware and the cost of low defect fabrication. To make these tradeoffs, methods must be developed to tolerate defects and faults, and to quantify costs and benefits.

A defect and fault tolerant computer is subject to many kinds of errors and requires support by both static techniques as well as dynamic reconfiguration. Soft errors are temporary in nature, and can be effectively handled through hardware-based redundancy techniques such as triple-modular redundancy (TMR), duplication-with-comparison (DWC), or concurrent error detection (CED). Hard errors are permanent, and result from degradation of devices over time. To mask these errors, hardware reconfiguration can move the application circuit from defective devices onto other functional areas of the chip. Reconfiguration can be done by the chip itself, or at higher levels such as in a Built-In Operating System (BIOS) or operating system (OS). Depending on the device technology used, the computer architect can apply

Table 1.1: Defect and Fault Tolerance at Different Levels. A FDT computer may implement defect and fault recovery support at multiple levels. Tradeoffs must be made when deciding which features to implement at each level.

Level	Description	Potential Solutions
System		OS or BIOS health monitoring, dynamic routing
Processor		reconfiguration, permanent fault diagnosis, health monitoring/repair, dynamic routing
Component	(e.g., EXE unit)	TMR, reconfiguration
Module	(e.g., Adder)	TMR, reconfiguration
Gate		NAND Multiplexing, Majority Voting
Device		Transistor reliability improvements

solutions at one or more levels, as shown in Table 1.1. These ideas are discussed in later chapters.

1.4.2 A Notional Architecture. This section presents a notional model of a FDT computer and shows how a reliable processor can be fabricated from an unreliable process technology. This general model will be expanded and used as the operating paradigm for the FDT computer. Figure 1.2 shows the steps in fabrication testing and operation of the processor.

Testing is done in two phases: fabrication and normal operation. After initial fabrication, each chip is tested and defects are located. The defects are evaluated to determine whether the chip can be used for the intended application. If too many defects exist, the chip is discarded. If not, the application circuit is mapped onto the device and placed into normal operation.

In normal operation, the chip continually monitors its health to detect errors. Soft errors are corrected using redundancy techniques. Permanent failures are corrected by reconfiguring the application circuit and reaccomplishing the calculation if necessary. In this way, the microchip can operate reliably, until the number and location of faults exceeds the chip's ability to recover.

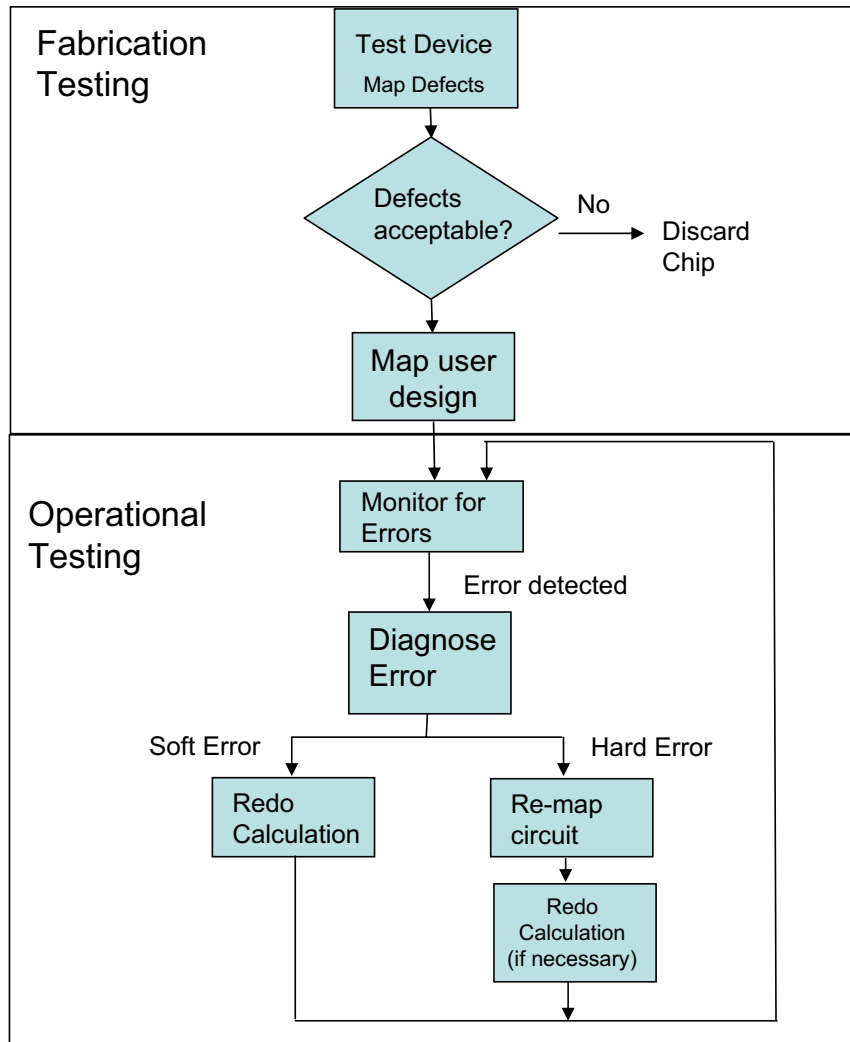


Figure 1.2: System Architecture of a FDT Computer. Steps in the initial test and operation of a defect and fault tolerant microprocessor.

1.5 *Summary*

The field of defect tolerant computing is seeing renewed interest. Developing an economical, efficient strategy at the device and architectural levels to use future device technologies is a laudable goal. Defect tolerant computing will be a valuable tool for the electronics industry in extending Moore's Law for the next several decades.

The remainder of this document presents a fault and defect tolerant computer architecture. Chapter II discusses the causes of the problem, challenges to be faced, and technologies that provide the basis for a solution. Chapter III defines three specific research goals and explains the contributions of this research. Chapter IV outlines the research methodology, and shows how the research goals were achieved.

Research results are presented in Chapters V through X. Chapter V proposes mathematical models for yield and hardware cost of the fault tolerance techniques used in the architecture. Chapter VI derives the first ever accurate analytical model for NAND multiplexing at moderate levels of redundancy. Chapter VII proposes a high level architecture for the fault and defect tolerant (FDT) computer. Chapter VIII develops the architecture for the FDT cache memory. Chapter IX develops the architecture of the remainder of the processor, and demonstrates the effectiveness of the fault tolerance strategies. Chapter X shows how these techniques can be implemented using non-silicon CMOS device technologies. Chapter XI presents conclusions and recommendations for future research. Appendix A includes additional background information on reconfigurable computing and programmable logic devices as well as information on two key technologies which may be used in defect tolerant computing. Finally, Appendix B proposes three follow-on phases for a long-term research program in this area.

II. Background

2.1 Overview

Overcoming the challenges of designing a defect tolerant computer requires the synthesis of several research areas. This chapter provides more detail on the problem area as well as supporting research which will be useful. This chapter is arranged as follows:

- Section 2.2 investigates problems associated with continued scaling of conventional silicon CMOS, potential solutions, and the impact on conventional computer architectures designed to use these devices.
- Section 2.3 introduces several device technologies that may eventually replace silicon CMOS. These devices, which include Single Electron Transistors (SETs), Quantum Cellular Automata (QCA), DNA self-assembled devices, and molecular crossbars, offer continued size scaling, but have significant limitations to overcome to achieve reliable computing.
- Section 2.4 summarizes Fault Tolerance (FT) techniques and typical applications. Fundamental FT approaches and methods are described, as are methods for modelling effectiveness.
- Section 2.5 examines radiation effects on CMOS devices.
- Section 2.6 summarizes current research in Fault Tolerant Architectures. Several researchers have proposed high level architectures for reliable computing with unreliable components.

2.2 Challenges in Nanometer Scale CMOS

Reduction in the size of silicon CMOS devices continues in accordance with Moore's Law as it has for the past forty years [Moo65]. The conventional planar silicon complementary metal-oxide-semiconductor (CMOS) process will continue to dominate the mainstream semiconductor market for at least the next ten years. As

devices continue to shrink, however, new challenges must be overcome. Quantum effects are beginning to dominate device performance, and conventional scaling rules that have worked well in the past are nearing their limits. As has happened in the past, pessimists are predicting the end of conventional CMOS in the foreseeable future. Scientists and engineers continue to devise new and innovative techniques to overcome these difficulties. This section examines some of the difficulties involved with scaling CMOS into the sub-100 nanometer range, potential impacts on yield and performance, and looks at several of the proposed solutions. Topics include short channel effects, quantum effects on device operation, fabrication problems due to process control and limitations in lithography, and other challenges. Proposed solutions include halo doping, silicon on insulator transistors, high-K dielectrics, new transistor designs, strained silicon and damascene interconnect.

2.2.1 The Silicon Roadmap. The term *nanometer scale CMOS* is typically used to refer to a CMOS process with feature sizes of less than 100 nanometers. Several definitions of process size exist. Historically, the process size has referred to the smallest feature that could be formed using lithography. In the past, this was also the transistor gate length. More recently, sublithographic processes have been developed that create gate lengths of one half the smallest lithographic feature size. Another definition is the size of a standard Dynamic Random Access Memory (DRAM) cell, which depends on the spacing between adjacent metal lines, called ‘pitch,’ as shown in the left diagram in Figure 2.1. For microprocessors (MPU) and Application Specific Integrated Circuits (ASIC), pitch definitions are based upon the distance between adjacent polysilicon or metal lines, as shown in the middle and right portions of Figure 2.1. The International Technology Roadmap for Semiconductors (ITRS) is published by a group of major semiconductor manufacturers. For convenience, the ITRS defines node size as the DRAM half pitch .

A typical N-type Metal-Oxide-Semiconductor Field Effect Transistor (MOS-FET) is shown in Figure 2.2. The MOSFET is a switch that controls the flow of

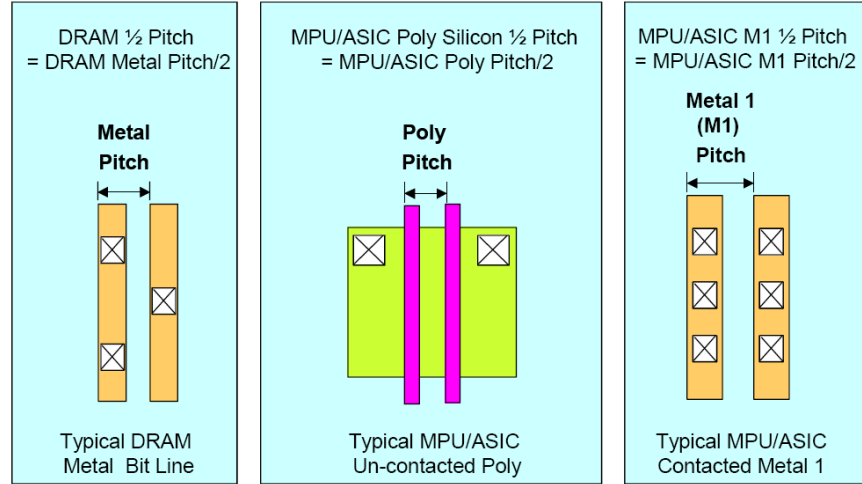


Figure 2.1: CMOS Node Size Definitions [Sem03]. The size of a transistor typically refers to the gate length. Pitch size is another useful definition.

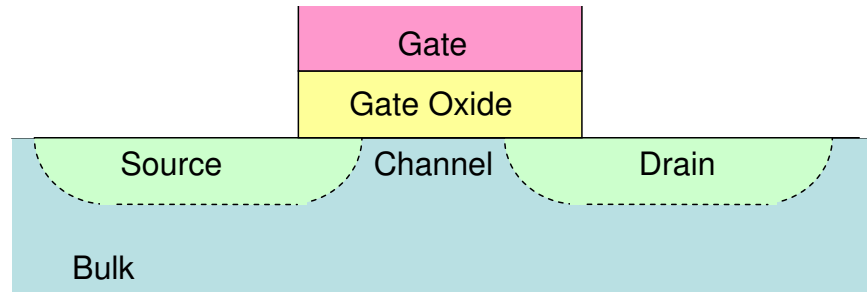


Figure 2.2: Cross Section of a Typical MOSFET.

current between the source and drain by modulating the voltage on the gate electrode. Fabrication begins with a wafer of intrinsic silicon, which is relatively free from impurities. The bulk region in which the transistor will be placed is doped with p-type impurities to create a region with an excess of electron holes. Next, a thin silicon dioxide (SiO_2) gate oxide layer is created between what will become the source and drain. A polysilicon or metal *gate* electrode is created on top of the gate oxide layer. Finally, the *source* and *drain* regions are created by adding n-type impurities to create regions with extra electrons. The region beneath the gate becomes the *channel*, through which current will flow when the transistor is turned on.

To operate the NMOSFET, a positive voltage is applied to the gate electrode. Although no current can flow between the gate and the channel region due to the

gate oxide insulator, a capacitive charge builds up as electrons in the bulk region are drawn up toward the Silicon-Silicon Dioxide boundary under the gate. After the threshold voltage is reached, sufficient numbers of electrons build up in the channel region to overcome the default p-type doping of the bulk and make the region n-type. This *inversion layer* between the source and drain forms a conductive channel through which current can flow. Removing the charge from the gate electrode allows the capacitive charge to drain from the channel and the electron concentration to return to the normal p-type levels, shutting off the flow of current.

The operating characteristics of the MOSFET are determined by the device dimensions, materials used, and dopant concentrations. To model device performance, *constant scaling rules* fix the ratios between dimensions and the rate of increase in dopant concentration as the overall device size shrinks. This process has worked very well for decades, but as process size shrinks into the nanometer regime, constant scaling rules no longer produce acceptable performance. Quantum effects and high electric fields become significant, and operation of the device changes. These problems must be overcome if scaling is to continue.

Each year, the ITRS establishes goals for the semiconductor industry by defining the desired process characteristics. The roadmap also identifies problems that must be overcome to achieve these goals, as well as current progress. A portion of the ITRS 2003 roadmap is shown in Table 2.1. Currently, the 90nm process, shown in the year 2004 column, is entering widescale usage. Lithographic limits are roughly 53nm, although sublithographic techniques allow physical gate lengths of 37nm. As gate length decreases, other dimensions must scale as well. Gate oxide thickness, T_{ox} , must shrink from 1.2nm down to 0.5nm. Dopant junction depth, X_j , must decrease from 30nm down to less than 11nm. Channel doping, N , must increase from $4 * 10^{18}cm^{-3}$ to over $2 * 10^{19}cm^{-3}$. Significantly, leakage currents, I_{off} , will continue to increase, raising static power consumption. Each of these parameters presents challenges. In fact, there are no known solutions to several of the problems identified in the roadmap. While researchers have always managed to find solutions to extend the roadmap, the

Table 2.1: ITRS Roadmap 2003 [Sem03]. Goals for CMOS characteristics are laid out by a semiconductor industry group. Many of the technologies needed to attain these goals have not been developed.

Year of Intro	2004	2007	2010	2013	2016
Node Size (nm)	90	65	45	32	22
Printed gate length L_g (nm)	53	35	25	18	13
Physical L_g (nm)	37	25	18	13	9
Max Power (W)	158	189	218	251	288
Transistors	193M	386M	773M	1.546G	3.092G
Clock Speed (GHz)	4.17	9.285	15.08	22.9	39.7
Interconnect Levels	10-14	11-15	12-16	12-16	14-18
Gate Oxide Thickness (nm)	1.2	0.9	0.7	0.6	0.5
Junction Depth r_j (nm)	30	20	15	11	
Channel Doping (cm^{-3})	$4 * 10^{18}$	$6 * 10^{18}$	$1.2 * 10^{19}$	$2 * 10^{19}$	
Power supply voltage V_{dd} (V)	0.9/1.2	0.8/1.1	0.7/1.0	0.6/0.9	0.5/0.8
I_{off} (nA/ μm)	50	70	100	300	500
I_{on} (mA/ μm)	1110	1510	1900	2050	2400

issues are becoming increasingly difficult, and it may ultimately become impossible or simply too expensive to continue to shrink silicon CMOS. Potential successors to silicon CMOS are discussed in Section 2.3. The remainder of this section discusses the specific problems that occur in scaling silicon CMOS, as well as some potential solutions.

2.2.2 Circuit Effects.

2.2.2.1 Short Channel Effects. Transistors in mass production are typically long channel devices; the effective channel length (L') is much larger than the maximum depletion region depth (W_m). A typical MOSFET is shown in Figure 2.3. The dotted lines denote the edges of the depletion regions formed at the PN junctions. As channel length decreases, the long channel circuit models used to predict performance begin to break down. Short channel effects include threshold voltage roll-off in the linear region, drain-induced barrier lowering (DIBL), and bulk punch

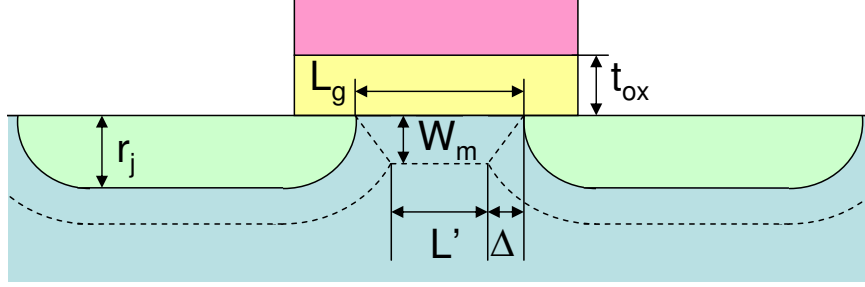


Figure 2.3: Detailed cross section of a typical MOSFET, showing several important parameters that determine device characteristics

through [Sze02]. The two main impacts are on threshold voltage and leakage current. Most of the solution strategies address one or both of these two effects.

2.2.2.2 Threshold Voltage. In theory, to reduce power consumption and increase speed, it is desirable to decrease both the threshold voltage, V_t , and the power supply voltage, V_{dd} . The threshold voltage cannot be reduced indefinitely because inverse subthreshold slope, a measure of the transistor turn-off rate versus gate voltage, is largely driven by thermally activated diffusion and is relatively independent of V_{dd} and channel length, L [TBC⁺97]. In addition, lower threshold voltage makes the transistors more susceptible to noise and ionizing radiation-induced logic errors (i.e., soft errors), and increases the off-state leakage current. The increase in leakage current is significant, increasing up to ten fold for every 0.1V reduction in V_t . A minimum V_t of 0.3-0.4V will likely be required to keep leakage currents at acceptable levels. Threshold voltage tends to roll-off as short channel effects become more significant due to charge sharing. The standard threshold voltage equation [Sze02, Eq. 6.45] is

$$V_t \approx V_{FB} + 2\psi_B + \frac{\sqrt{2\varepsilon_s q N_A (2\psi_B + V_{BS})}}{C_o} \quad (2.1)$$

where V_{FB} is the flat band voltage, Ψ_B is the electrostatic potential difference between E_F , the Fermi Energy, and E_i , the intrinsic energy, and C_o is the oxide capacitance. N_A is the substrate doping concentration, q is the elementary electron charge, ε_s is

the permittivity of silicon, and V_{BS} is the potential difference between the bulk and source. Drain-Induced Barrier Lowering (DIBL) reduces V_t and increases the sub-threshold current. Ultimately, the gate loses control over the channel. The reduction in threshold voltage can be shown [Sze02, Eq. 6.47] as

$$\Delta V_T = -\frac{qN_A W_m r_j}{C_o L} \left(-1 + \sqrt{1 + \frac{2W_m}{r_j}} \right) \quad (2.2)$$

where W_m is the junction depth in the channel, r_j is the junction depth in the source and drain, and L is the channel length. To keep the shift in V_t small, a designer can decrease W_m , but this will ultimately create an increase in leakage current due to quantum tunnelling between the gate and the substrate. The junction depth, r_j , can be reduced, but this will also increase leakage currents due to tunnelling between the source/drain and substrate. Another option is to increase the gate capacitance, C_o . All of these options have their own challenges. Good transistor design requires careful balance of dimensions and doping levels to achieve good performance at the lowest feasible operating voltage and with the smallest leakage currents. Balancing these parameters is becoming more challenging as device size shrinks. Careful control of the threshold voltage is important lest the device no longer function as a transistor.

2.2.2.3 Constant Scaling Rules. The usual method of avoiding short channel effects as gate length decreases is to scale other device parameters as well, according to a set of *constant scaling rules*, as shown in Table 2.2. Device dimensions are scaled relative to each other, which keeps the electric fields as they would be in a long channel device. This technique has worked well in the past, but is beginning to approach limits. In particular, as device dimensions decrease, problems begin to arise due to increasingly high dopant concentrations required in the channel, and quantum tunnelling due to the high electric fields in the source and drain, very shallow junction depths, and thin gate oxide thicknesses.

Table 2.2: Constant Field Scaling Rules [Sze02]. As the transistor size shrinks, many other factors scale as well, often with negative consequences.

Determinant	MOSFET device and circuit parameters	Multiplying Factor ($\kappa > 1$)
Scaling Assumptions	Device dimensions (d, L, W, r_j) Doping concentrations (N_A, N_D) Voltage (V)	$1/\kappa$ κ $1/\kappa$
Derived scaling behavior of device parameters	Electric field (ε) Carrier velocity (v) Depletion layer width (W) Capacitance ($C = \varepsilon A/d$) Inversion layer charge density (Q_n) Drift current (I) Channel resistance (R)	1 1 $1/\kappa$ $1/\kappa$ 1 $1/\kappa$ 1
Derived scaling behavior of circuit parameters	Circuit delay time ($\tau \sim CV/I$) Power dissipation per circuit ($P \sim VI$) Power-delay product per circuit ($P\tau$) Circuit density ($\sim 1/A$) Power density (P/A)	$1/\kappa$ $1/\kappa^2$ $1/\kappa^3$ κ^2 1

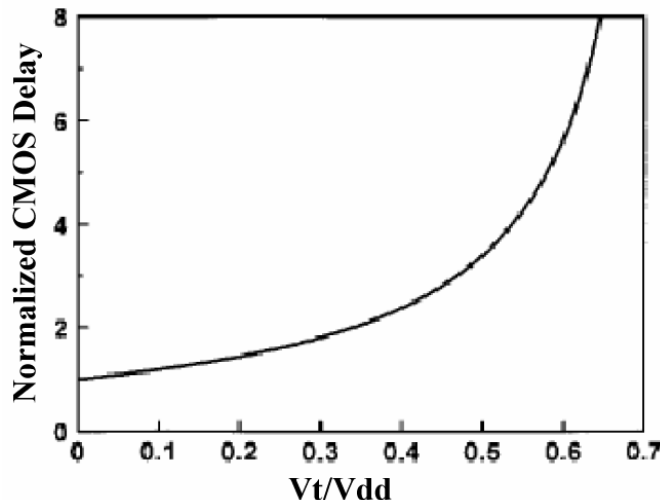


Figure 2.4: Normalized Delay vs. V_t/V_{dd} Ratio of a CMOS inverter [TBC⁺97]. Transistor switching speed tends to slow as the device size gets smaller.

2.2.2.4 Power Supply Voltage. One element that cannot be scaled in accordance with the constant scaling rules is power supply voltage, V_{dd} . To reduce the susceptibility of the devices to noise, it is desirable to maintain V_t even as the power supply voltage decreases. Thus the ratio of V_{dd} to V_t goes down as process size shrinks. This decreases the switching speed of the transistors and their ability to drive other devices. Figure 2.4 shows the normalized simulated delay per stage of a 1.5V, 0.1 μm CMOS inverter versus the V_t/V_{dd} ratio. The delays are normalized to 10 ps at $V_t/V_{dd} = 0$. As the threshold voltage approaches the power supply voltage, the delay of the inverter increases parabolically.

2.2.2.5 High Field Effects. A consequence of maintaining a higher V_{dd} than specified in the constant scaling rules is an increase in the electric fields in the device. For a 0.1 μm device, the electric field in the gate oxide can be greater than 5MV/cm, and greater than 1MV/cm in the silicon substrate. Such severe bending of the energy bands in a small distance allows quantum tunnelling between the source/drain and substrate, which leads to an increase in leakage current. In addition, electron mobility decreases under high electric fields due to phonon scattering

and surface roughness at the $Si-SiO_2$ interface. Reduction in mobility reduces device switching speed.

2.2.2.6 Quantum Effects on Device Operation. As device sizes shrink, the current density in the inversion layer increases as well, increasing almost ten orders of magnitude as the gate oxide thickness decreases from 36 angstroms to 15 angstroms [TBC⁺97]. As the gate oxide thickness decreases, quantum tunnelling begins to occur between the gate and the substrate. Under these conditions, the electrons in the inversion layer can be treated as a 2D electron gas. Electrons in the inversion layer occupy discrete energy levels. In device operation, the lowest energy level in the conduction band is above E_C , the normal edge of the conduction band for the channel layer (with no applied gate voltage). Higher potential is necessary to bring electrons into the conduction band, thus increasing the threshold voltage of the transistor.

2.2.2.7 Gate Oxide Thickness. As devices shrink, the required gate oxide thickness decreases as well. Gate-Source current leakage can be appreciable whenever the source is biased, whether the gate is biased or not. In addition, quantum tunnelling effects create a leakage current between the gate and substrate and the gate and drain. For a $0.1\mu m$ MOSFET at 1.5V, the gate oxide thickness should be 30 angstroms [TBC⁺97], which is only 10 atoms of silicon. A further decrease in gate oxide thickness causes leakage current to grow exponentially. Below about 20 angstroms, oxide tunnelling current becomes unmanageable [TBC⁺97]. This limits channel length to roughly 25-50 nm, a limit that is already being reached. One solution to this problem, the replacement of the silicon dioxide dielectric with alternate materials, is discussed in Section 2.2.3.4.

2.2.2.8 Dopant Fluctuations. As devices get smaller, the number of dopants in the channel can drop to the level of only hundreds. Statistical averages no longer apply, and variations in dopant concentrations from device to device (or from

run to run) can have significant effects on threshold voltage, device speed, and circuit drive capability. The standard deviation of the threshold voltage due to variations in the number of channel impurities is

$$\sigma_{V_t} = \frac{q}{C_{ox}} \sqrt{\frac{N_A W_{dm}}{3LW}} \left(1 - \frac{x_s}{W_{dm}}\right)^{3/2}, \quad (2.3)$$

where C_{ox} is the oxide capacitance, N_A is the substrate doping concentration, W_{dm} is the maximum channel depletion region depth, L is the channel length, W is the channel width, and x_s is the thickness of an undoped surface layer under the gate [TBC⁺97]. The x_s parameter comes from a proposed solution, known as retrograde channel doping, discussed in Section 2.2.3.2.

2.2.2.9 Interconnect Delay. As devices shrink, interconnect delay becomes an increasingly large fraction of the overall circuit delay. The relative delays of logic gates, local interconnect, and global interconnect is shown in Figure 2.5 [Sem03]. As process size shrinks, gate delay and local interconnect delay becomes a smaller fraction of overall delay. Global interconnect delay comes to dominate, although the effect can be reduced through the use of repeaters to increase driving current over long distances across the chip.

Wire propagation delay, τ , is

$$\tau = RL_w \left(C_L + \frac{1}{2}CL_w \right), \quad (2.4)$$

where R is resistance per unit wire length, C is capacitance per unit wire length, L_w is wire length, C_L is load capacitance [TBC⁺97]. As device sizes shrink (i.e., L_w gets smaller), R increases due to the smaller cross section of the wires, while C and C_L usually stay constant. To keep the delay small, the RC time constant must be reduced.

One method is to reduce the wire capacitance, C , through the use of low K dielectrics, such as polyimide or F-doped SiO_2 , around metal lines. Another method

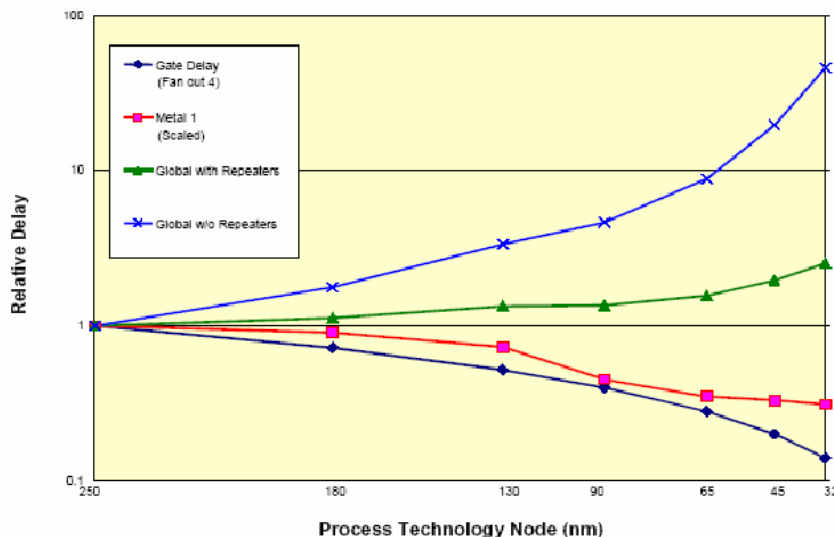


Figure 2.5: Increasing Effects of Interconnect Delay [Sem03].

uses metal with a lower resistivity than aluminum, such as copper. A damascene process is used in this case, and is similar to that used for metal gates (cf., Section 2.2.3.4). Finally, resistance in the wires can be reduced by increasing their cross sectional thickness, mostly for the case of upper metal levels used for global interconnect. While in the past clock speed has been largely limited by device switching speed, interconnect delay will become more of a problem in the future. Some designs already require more than one clock cycle to send a signal a long distance across the chip. Interconnect delay will be an important design consideration in the future.

2.2.2.10 Radiation Effects. As process size shrinks, the devices become more prone to errors caused by the impact of alpha particles and high energy neutrons, electrical noise, as well as fluctuations in the power supply voltage. These effects are discussed further in Section 2.5.

2.2.3 Potential CMOS Solutions. This section examines several proposed solutions to the problems discussed in Section 2.2.2.

2.2.3.1 Lithography. Lithography is the process used to create features during fabrication. Fabrication of a microchip is done in several stages, starting with an initial wafer of silicon. At each stage, light is shined through a mask (similar to a photographic negative) to create the patterns that define the circuit structure. These patterns determine where impurity doping is done, as well as where metal lines and polysilicon gates are formed. Optical lithography has two main challenges: accurate placement and alignment of the masks, and minimum feature size.

Accurate placement of the masks is vital to creating a functional circuit. A fabrication process can involve more than twenty masks, each of which must be aligned on the microchip with extremely high precision. Misalignment of the mask can easily result in the destruction of the entire wafer. As the process size shrinks, the masks must be aligned with ever greater precision. Current mask aligners must be carefully shielded from vibration and are becoming increasingly expensive.

The other challenge facing designers is limitations of optical lithography. The smallest feature that can be produced is typically a function of the wavelength of light used. Below this resolution, features of the mask become smeared due to diffraction. VLSI designers create design rules at each process size that specify how large features must be and how closely lines can be placed for fabrication. As the process size shrinks, the number of rules has increased into the hundreds, greatly increasing the complexity of VLSI design.

Modern production systems use a KrF excimer laser to produce light at $\lambda = 248\text{nm}$. Research systems using ArF lasers at 193nm allow resolutions down to 180nm. Few expect optical lithography to be useful for sub-100nm processes [TBC⁺97], although hybrid techniques combining optical lithography with chemical etching and other techniques may allow features below 100nm. Techniques such as near-field phase correction, ashing-trimming, and the use of sacrificial layers allow patterning of features smaller than the wavelength of light used. One sublithographic patterning technique is illustrated in Figure 2.6 [CCH⁺03]. Parts (a) and (b) illustrate how the

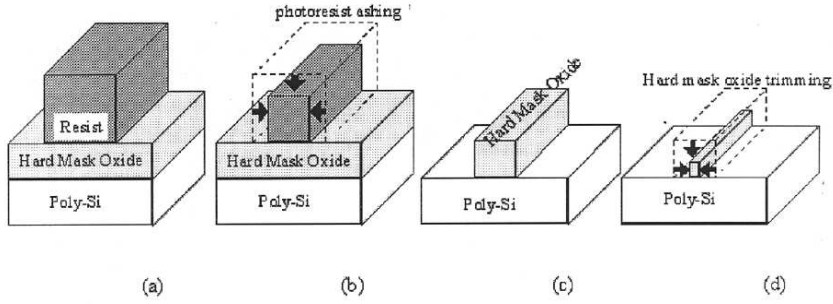


Figure 2.6: Sublithographic Patterning enables creation of feature sizes smaller than the wavelength of the light used in lithography [CCH⁺03].

photoresist pattern is decreased in thickness by oxide plasma ashing (i.e., burning away part of the resist). Parts (c) and (d) illustrate how the oxide itself is trimmed via chemical etching using hydrofluoric acid. As cited in [CCH⁺03], these techniques have been used to reduce 500nm line widths down to below 20nm [ACKH01].

Possible replacements for optical lithography are X-Ray lithography, parallel electron beams, extreme ultraviolet lithography, and ion beams. The most promising technology is likely using X-rays. Figure 2.7 shows 80nm lines patterned using synchrotron X-Ray lithography. Research in near-contact X-ray lithography shows features can be patterned to 30nm. This should be sufficient for MOSFETs down to the perceived limits in size. The biggest challenge to overcome for X-ray lithography is the fabrication of the mask. Thin membranes (2-5 μm) of silicon or silicon compounds such as Si_3N_4 or SiC must be patterned with an X-ray absorbing material such as gold. Whereas optical lenses allow the reduction in size of mask features, X-ray masks must be fabricated at a 1:1 ratio. As a result, the mask has an extremely low tolerance for defects. In addition, the mask must be placed very close to the wafer surface, requiring very flat masks and planar wafer surfaces.

Another potential technology is electron beam lithography, used in the research environment for many years. The challenge in this case is throughput, as the Gaussian probes used can be focused on a spot of only a few nanometers at a time [TBC⁺97].

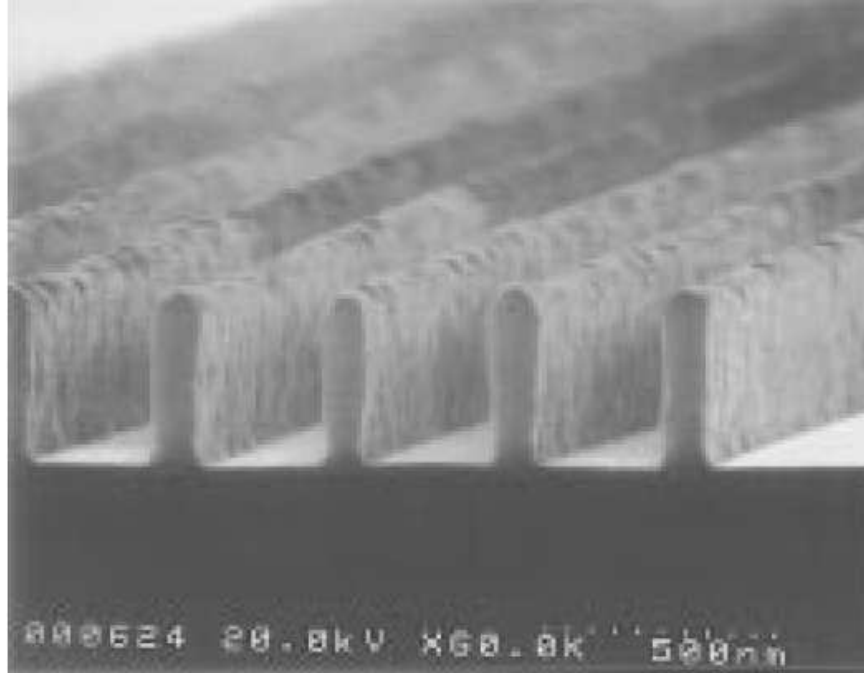


Figure 2.7: 80nm Lines Patterned with X-Ray Lithography [TBC⁺97]. The mask wafer gap was $25\mu m$.

2.2.3.2 Process Technology. Several process technology improvements have recently been shown to decrease the variation of threshold voltage and other parameters. This section discusses the use of *halo doping*, *retrograde channel doping*, *Thin Film Transistors (TFT)* and *Silicon On Insulator (SOI) transistors*.

Halo Doping Halo doping reduces the effect of the source and drain electric fields on the channel region. This allows the gate to more effectively control the channel. As shown in Figure 2.8 , pockets of high doping are introduced at the edges of the source and drain at the time of source/drain doping via ion implantation. These regions are only partially depleted during device operation, and thus shield the channel from further penetration of the high fields. The depletion regions for a conventional MOSFET were shown by dotted lines in Figure 2.3. The effective reduction in channel length caused by encroachment of the source and drain depletion regions into the channel is shown by Δ . Halo doping reduces this value. One problem with halo doping is that rapid thermal annealing, one

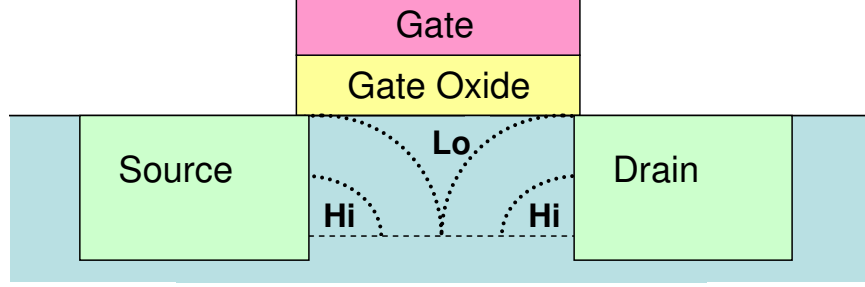


Figure 2.8: Halo Doping is one technique used to minimize the short channel effects in silicon CMOS [TBC⁺97].

of the typical steps in the fabrication process, is made more difficult as care must be taken not to allow the halo doped areas to diffuse into the surrounding silicon.

Retrograde Channel Doping Retrograde channel doping is under investigation for use below $0.2\mu m$. It employs non-uniform doping in the vertical direction to reduce short channel and impurity concentration effects on threshold voltage. Figure 2.9 shows an example band diagram and doping profile. No doping is introduced in the channel at shallow depths, but begins at $x_s = W_{dm}$, the maximum depletion region depth. The band diagram at the top shows the FET at the onset of inversion. By altering the doping profile, the energy bands are bent such that more electrons enter the inversion layer at the channel surface at a lower voltage. As a result, threshold voltage is reduced. For the same gate depletion width, W_{dm} , the surface electric field and the total depletion charge for the retrograde channel is one half that of a uniformly doped channel [TBC⁺97].

The second benefit of retrograde channel doping is that threshold voltage variation due to impurity concentrations can be reduced. The equation for the standard deviation in threshold voltage due to channel doping is

$$\sigma_{Vt} = \frac{q}{C_{ox}} \sqrt{\frac{N_A W_{dm}}{3LW}} \left(1 - \frac{x_s}{W_{dm}}\right)^{3/2}. \quad (2.5)$$

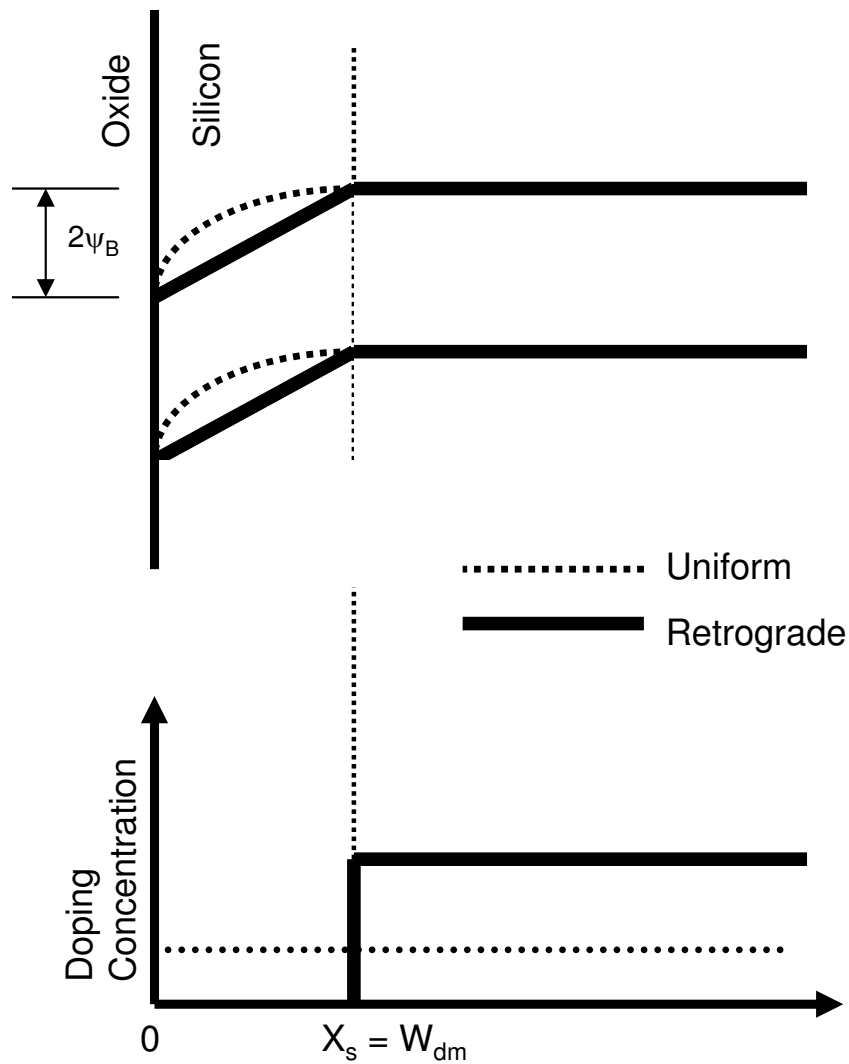


Figure 2.9: Retrograde channel doping reduces the effects of impurities on threshold voltage [TBC⁺97]. Energy band diagram is shown at the top of the figure, and doping concentration is shown at the bottom.

Note that when $x_s = W_{dm}$ as in the retrograde case, the term goes to zero. Thus, the effects of process variation on threshold voltage due to impurity concentration can be removed.

Silicon On Insulator Much research has been done on silicon on insulator (SOI) and thin-film transistors (TFT). Originally, SOI was used for radiation-hardening applications, but is increasingly under investigation for general purpose applications. SOI can be grouped into two types: fully-depleted SOI (FD-SOI) and partially-depleted SOI (PD-SOI). In FD-SOI, the depletion region of the source and drain goes to the bottom of the silicon layer to the underlying insulator. In PD-SOI, it does not. In the PD-SOI case, care must be taken to avoid the floating body effect, which results in a potentially significant leakage current between the source and drain due to a forward-biased body-to-source junction when V_{ds} is high [TBC⁺97, page 497]. This effect can be eliminated through proper layout and the use of body contacts.

2.2.3.3 Device Design. In addition to process and material improvements, research is being conducted into new transistor designs. These designs attempt to overcome limitations in lithography and allow more effective control of the channel. Two research areas are vertical transistors, and two/three-gate transistor designs.

Vertical Transistors. In the vertical transistor, channel length is controlled by epitaxy instead of lithography (Figure 2.10). FETs with effective gate lengths of less than 4 nm have been fabricated in labs [Ris02]. One challenge to be overcome is that of large leakage currents.

Two/Three Gate Transistors. Three types of double-gate FETs (DGFETs) are under investigation. The main advantage of multi-gate designs is to reduce the effect of the drain field on the channel, mitigating short channel effects. For the same channel thickness, the DGFET can be scaled to approximately 2-3 times

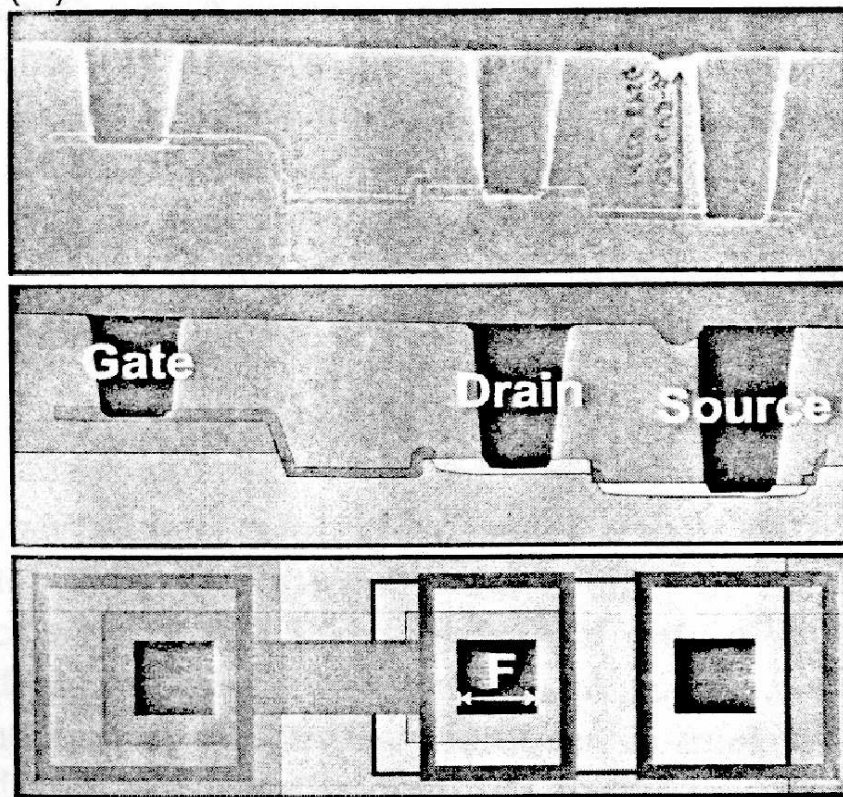


Figure 2.10: Vertical Transistor [Ris02]. Alternative transistor designs have been proposed to overcome some scaling problems.

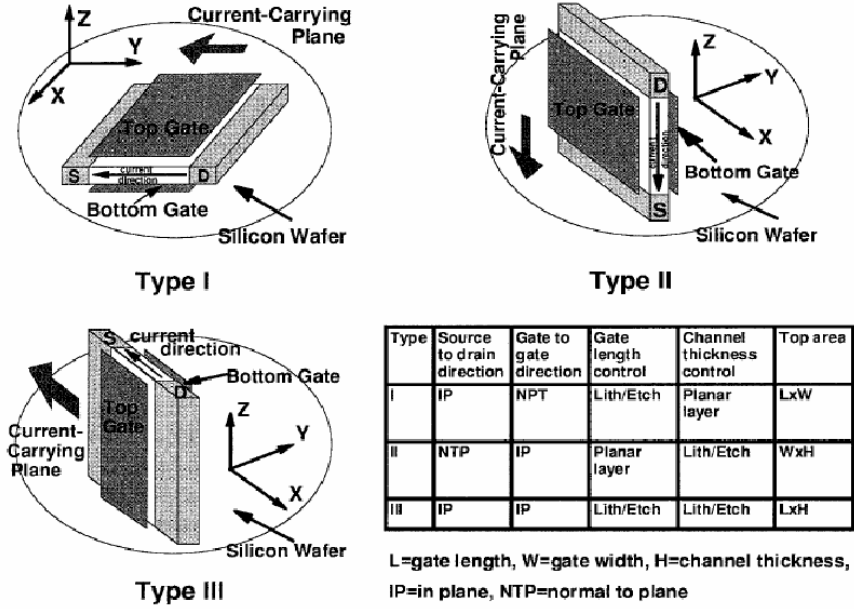


Figure 2.11: Two Gate FET Designs [Won02] provide more efficient control of current flow.

shorter channel length [Won02]. Another benefit is that threshold voltage variation between devices is smaller than for single-gate FETs.

As shown in Figure 2.11, the primary difference between the designs is in orientation. Each of three device structures has advantages. In the figure, ‘S’ and ‘D’ represent the source and drain, respectively. Type I devices have a horizontal channel, whose thickness is controlled by epitaxy rather than lithography. Small thicknesses are more readily controlled via epitaxy than lithography. Type II and III devices have the most compact footprint, and are under investigation for DRAM applications. In all of these devices, fabrication is more difficult than for conventional planar FETs. Aligning the two gates above the channel is much more difficult than for the single gate self-aligned process.

2.2.3.4 Materials. Replacement of the traditional materials used in conventional CMOS design with more advanced materials has been shown to reduce the negative effects of scaling. This section discusses *Strained silicon*, *High K dielectrics*, *metal gates*, and *damascene interconnect*.

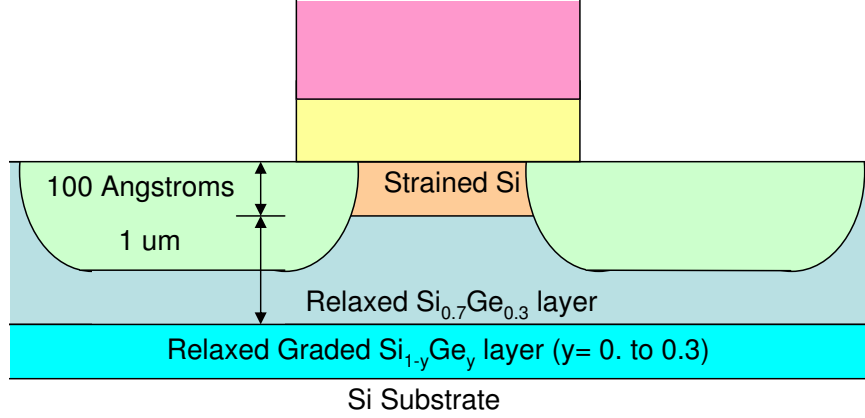


Figure 2.12: Typical NMOS Strained Silicon FET. Strained Silicon MOSFETs are currently entering production [HNE⁺02].

Strained Silicon. Strained silicon is already used by Intel [Gep02, HNE⁺02, WHG92, WWC⁺03, Boh03, GAA⁺03]. With this technique, electron and hole mobility is increased by inducing tensile strain or compression in different silicon layers using various alloys of Si_xGe_{1-x} , as illustrated in Figure 2.12. Typical alloy compositions are from 10-30% Ge, although some attempts have been made to make PMOSFETs out of 50% Ge [TBC⁺97, HNE⁺02]. A variety of different alloy compositions and structures have been reported [Gep02, Ser03, TBC⁺97, HNE⁺02, WHG92, WWC⁺03, Boh03, GAA⁺03].

The typical architecture for a strained-silicon NMOSFET induces a strain in the silicon channel layer. Germanium atoms replace some of the silicon atoms below the surface. A thin layer of Silicon is grown on top of the Si-Ge layer. Since Ge is larger than Si, the lattice constant of the Si-Ge layer is correspondingly higher than the Si layer. This creates a lattice mismatch between the Si and Si-Ge layers. The Si layer is thus strained in the horizontal direction, and compressed in the vertical. IBM researchers report an increase in electron mobility of over 60% [Gep02], with 25-30% improvements in device speed.

Mobility is increased due to a change in the energy band structures. The Si atoms are further apart in the horizontal direction than in unstrained Si, which reduces

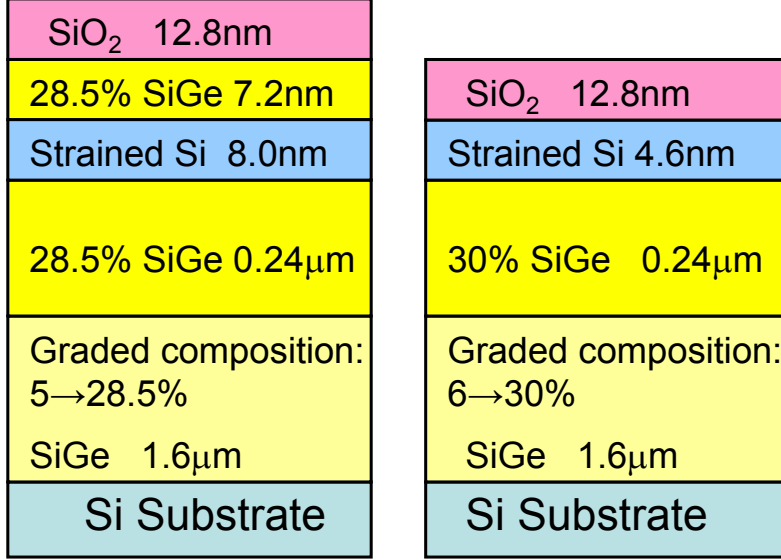


Figure 2.13: PMOS and NMOS Strained Silicon Structures [WHG92]. The left diagram is a buried-strain device used for PMOS, while the right diagram is a surface-strained device used for NMOS.

the number of collisions with phonons [Gep02, page 30]. Two alternative designs for PMOS and NMOS FETs are shown in Figure 2.13. For a PMOS device, the atoms should be compressed to increase hole mobility. Researchers have done this by compressing the Si layer externally [Boh03,GAA⁺03], or by using the SiGe layer as the hole channel, as shown in the left part of the figure [TBC⁺97]. For a NMOS device, the strained silicon layer is used for the channel, as shown in the right side figure. In an oxide-gated sample, with 70 angstroms oxide thickness and a buried strained-Si channel, high electron mobilities in excess of $2200\text{cm}^2/\text{V-sec}$ can be maintained at an electron density of $3 * 10^{12}\text{cm}^{-2}$ [TBC⁺97, page 497] (versus the more typical $1450\text{cm}^2/\text{V-sec}$ for unstrained silicon). Hole mobilities in excess of $800\text{cm}^2/\text{V-sec}$ can be maintained (versus $505\text{cm}^2/\text{V-sec}$ for unstrained).

Intel has recently introduced another method [Boh03,GAA⁺03] for use in 90nm CMOS processing, as illustrated in Figure 2.14. In their approach, strain is created in the NMOS channel by using a 75nm silicon nitride (Si_3N_4) capping layer on top of the NMOSFET to create a tensile strain on the channel layer underneath(Figure 2.14).

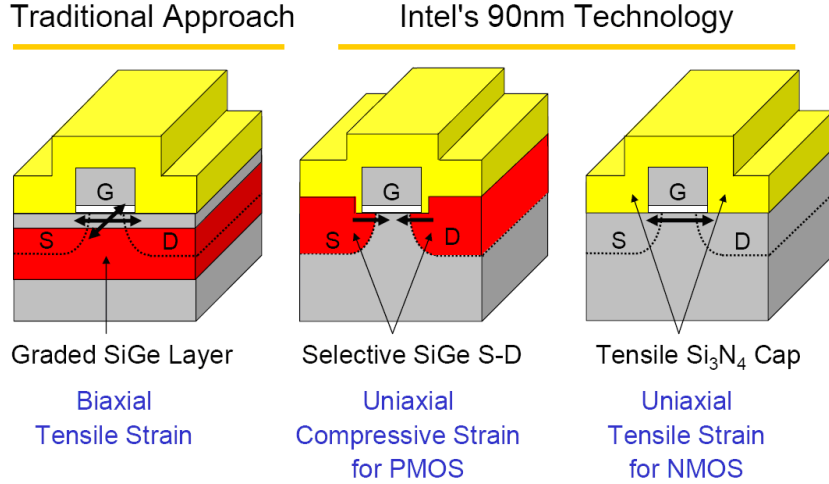


Figure 2.14: Intel's Strained Silicon Approach [Boh03].

For PMOS devices, selective epitaxial growth of SiGe is done for the source and drain regions, which create a compressive strain on the silicon channel layer. Intel reports a greater than 50% hole mobility increase for a 17% Ge composition [GAA⁺03].

Strained silicon presents several challenges in fabrication. The growth conditions used must be optimized for strained materials. Intel's approach requires selective SiGe epitaxy. Care must be taken to avoid strain relaxation, such as the restriction to low temperatures ($< 800K$) in later processing steps. In addition, scaling limits may become a significant factor, as layer thicknesses must be greater than roughly 30 angstroms lest quantum effects become significant.

High K Dielectric Materials. High gate capacitance is a key factor in controlling the flow of current through the channel, threshold voltage, and leakage currents. Obtaining high gate capacitance requires a very thin oxide layer, which increases quantum tunnelling and current leakage. Replacement of silicon dioxide with alternative materials can provide the same capacitance with thicker layers. Gate capacitance is

$$C = \frac{\epsilon A}{d} \quad (2.6)$$

Table 2.3: Properties of Selected High K Dielectrics [Won02].
The high dielectric constants of these materials allows the use of thicker gate oxide layers. Unfortunately, many of the materials have problems with high leakage current relative to silicon dioxide.

Dielectric	Dielectric Constant (bulk)	Bandgap (eV)	Conduction band offset (eV)	Leakage Current w.r.t SiO_2	Thermal stability w.r.t Si
Silicon Dioxide (SiO_2)	3.9	9	3.5	N/A	$> 1050C$
Silicon Nitride (Si_3N_4)	7	5.3	2.4		$> 1050C$
Aluminum Oxide (Al_2O_3)	~ 10	8.8	2.8	$10^2 - 10^3 \times$	$\sim 1000C$, RTA
Tantalum Pentoxide (Ta_2O_5)	25	4.4	0.36		Not thermodynamically stable with Si
Lanthanum Oxide (La_2O_3)	~ 21	6	2.3		
Hafnium Oxide (HfO_2)	~ 20	6	1.5	$10^4 - 10^5 \times$	$\sim 950C$
Zirconium Oxide (ZrO_2)	~ 23	5.8	1.4	$10^4 - 10^5 \times$	$\sim 900C$

where A is the gate area, d is the oxide thickness, and ε is the electrical permittivity of the oxide [Sze02]. As the device area is scaled smaller, there are two options to maintain gate capacitance. The oxide thickness could be decreased as well, which allows quantum tunnelling and increased leakage currents. In addition, a material with a higher permittivity can be used in place of the silicon dioxide. Table 2.3 shows several materials under investigation for this purpose.

The use of a material such as tantalum pentoxide (Ta_2O_5), with a dielectric constant of 6.4 times that of silicon dioxide ($\varepsilon_t = 6.4\varepsilon_s$), allows a oxide layer (d_t) that is 6.4 times as thick to create the same gate capacitance (C_{ox}), while reducing the effects of quantum tunnelling. This relationship is shown by

$$C_{ox} = \frac{\varepsilon_s A}{d_s} = \frac{\varepsilon_t A}{d_t} = \frac{6.4\varepsilon_s A}{6.4d_s}. \quad (2.7)$$

Intel is investigating the use of Zirconium oxide (ZrO_2) [Ser03]. It is made by depositing $ZrCl_4$ onto the wafer, and then introducing steam. One disadvantage of using these materials is the requirement for lower temperatures in later processing steps. Much research remains to be done before these materials can be incorporated into mass production.

Metal Gates. Another potential technology is the use of metal for the gate instead of the more conventional doped polysilicon (poly). This is beneficial for several reasons: first, boron used in the doping of the polysilicon gate can penetrate into the channel, unintentionally introducing dopants into the channel and affecting the threshold voltage; Second, metal gates create an extremely small depletion region compared to poly. With poly, this depletion region adds to the parallel plate distance in the gate capacitance equation, reducing the capacitance. The smaller depletion region due to the metal gate allows a thicker gate oxide, reducing quantum tunnelling and leakage current. Finally, lower fabrication temperatures can be used versus those for poly deposition. This is desirable when high-K dielectrics are used, which can break down at higher temperatures.

Choices for the gate metal include aluminum, tungsten, molybdenum, ruthenium-tantalum alloys, and SiGe and SiC alloys. The big differences between the metals are in the work functions, which will in part determine the threshold voltage of the transistors. The NMOS and PMOS devices should ideally have gate metals with different work functions, or at least one that is approximately mid-gap for both [WFS⁺99, page 551].

Several challenges must be overcome to use metal gates. One of the most important is the potential for reactions with underlying oxides, which can be overcome through the use of a damascene gate process. An example of a damascene process

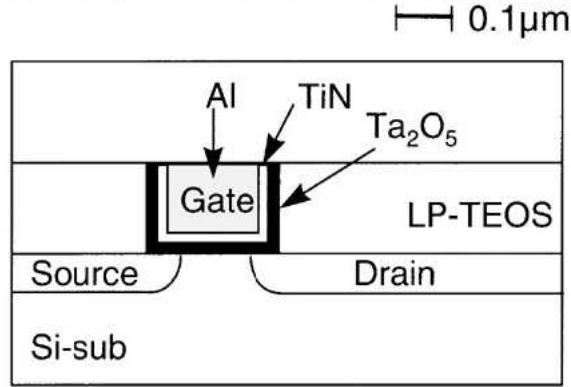


Figure 2.15: Damascene Gate Cross Section [YSN⁺00].

is shown in Figure 2.15. In this process, Ta_2O_5 is used as the gate oxide, while aluminum is used as the primary gate metal. Titanium Nitrate (TiN) is used to line the tub where the gate aluminum will be placed. This separates the two materials, and ensures the aluminum does not react with the Ta_2O_5 or leach into the channel below.

Steps in the fabrication process of a metal gate FET follows, and are shown graphically in Figure 2.16.

1. After shallow trench isolation is performed, a dummy gate oxide and dummy gate are grown using Si_3N_4 and polysilicon (Diagram one in the figure).
2. Sidewalls are grown from Si_3N_4 (Diagram 2).
3. Ion implantation is used to form the source and drain (Diagram 2).
4. A Pre-Metal Dielectric (PMD) film is deposited and polished using chemical-mechanical polishing (CMP) (Diagram 3).
5. Wet etching is done using hot H_3PO_4 , then chemical dry etching is used to remove the dummy gate and gate dielectric. Care is taken to avoid etching the sidewalls or Si substrate (Diagram 4).
6. A new gate oxide (SiO_2 or Ta_2O_5) is grown (Diagram 5).
7. TiN or similar metal is used as a barrier or glue metal (Diagram 5).

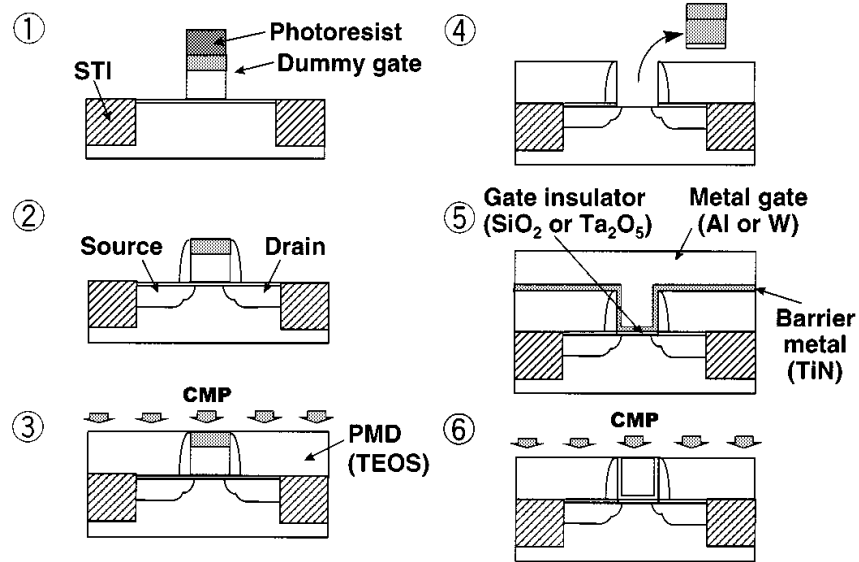


Figure 2.16: Damascene Gate Fabrication Process [YSN⁺00].

8. The gate electrode of W or Al is deposited using low-pressure chemical vapor deposition or DC magnetron sputtering (Diagram 5).
9. Chemical mechanical polishing is used to planarize the final structure (Diagram 6).

2.2.4 Limitations of the Current Design Paradigm. As discussed in the previous sections, continued scaling of silicon CMOS will be difficult. This section discusses the impacts of these problems at the system and economic levels. The following effects are already being observed:

- Variations in Device Performance,
- Increased Power Consumption,
- Lower Yield,
- Increased Cost, and
- Increased Soft Error rates.

Variations in Device Performance. As discussed earlier, smaller devices are constructed from fewer atoms. Small variations in dopant concentrations can have a large effect on the performance of each transistor. Variations in doping can cause the transistor to switch slower than other devices, decrease its ability to drive other devices, or cause it not to function at all. Testing the circuit for correct operation is becoming increasingly difficult and time consuming. Parametric errors caused by slow transistors can be very difficult to detect as they are often only observed on the worst case propagation paths. The end result of variations in doping is a decrease in production yield and overall reliability.

Increased Power Consumption. The quantum tunnelling effects and resulting leakage currents greatly increase the static power dissipation of the transistors. In the past, MOSFETs had very low static power dissipation, only dissipating appreciable amounts of power while switching. This will not be the case with smaller devices. Thus circuits will dissipate power whenever they are powered, regardless of whether they are being used. Designers will be under pressure to reduce the power supply voltage and threshold voltage to reduce power dissipation. This will make devices more susceptible to soft errors.

Even when not directly affecting reliability, power consumption must be a key concern in the design of fault and defect tolerant circuits. One of the primary methods of fault tolerance is redundancy via hardware duplication. The designer must account for the power consumed by redundant hardware.

Lower Yield. In a simple reliability model,

$$P_{functional} = (1 - P_{def})^N, \quad (2.8)$$

the reliability of the entire processor depends on the reliability of each transistor in the device. The challenges described in this section will reduce the probability that each transistor will work reliably.

Increased Cost. In the past, tremendous effort has been expended to tightly control the fabrication process. Current fabrication facilities cost roughly \$2 billion. Each successive generation of fabrication facility costs two to three times more than the last. Faced with this increasing cost, designers may be forced to accept less controlled processes, with higher defect rates. In this case, designers will be forced to adopt defect tolerance even for use with conventional silicon CMOS.

Soft Errors. Even if nanoscale CMOS devices can be fabricated with low defect rates, they will still be subject to increasing rates of soft errors caused by solar radiation and electrical noise. Many current memory devices already include parity bits or error correcting codes to allow correction of single event upsets (SEUs) that flip the bit value of memory bit. Currently, commercial processors do not need to detect or correct SEUs occurring in logic and its associated register memory. Research efforts in this field are discussed in Section 2.6.2.

The challenges facing the continued reduction in CMOS process size into the nanometer regime are significant. Constant scaling rules that have worked well for the last thirty years can no longer be used due to the increasingly significant effects of quantum tunnelling. A variety of solutions are under development, which may extend the utility of conventional planar silicon CMOS for the next decade. Even with these advances, low defect rates will be difficult to achieve, and operational devices will be subject to increasing numbers of soft errors. Whereas system reliability has been obtained primarily at the device level for many years, the time will come when computer architects must build reliability into their designs at the architectural level.

2.3 New Device Technologies

In the long term, new device types are likely to replace silicon CMOS. This is not unexpected, as silicon CMOS is but one of at least four device technologies used to implement digital logic (i.e., relays, vacuum tubes, bipolar junction transistors, and now field effect transistors). Indeed, with more than forty years of use, silicon CMOS has had the longest life of any of these technologies. Advancement of each technology

Table 2.4: Emerging Logic Device Families [HBZB02]. Several technologies are under investigation as replacements to silicon MOSFETs.

Device	Single Electron Transistor	Rapid Single Quantum Flux Logic	Quantum Cellular Automata	Carbon Nanotube Devices	Molecular Devices
Types	3-terminal	Josephson Junction & Inductance Loop	Electronic QCA Magnetic QCA	FET	2-terminal & 3-terminal
Advantages	Density, Power, Function	High Speed, “Potentially Robust”	Density, No Interconnect in signal path, Speed, Power	Density, Power	Density, Potential Interconnect Benefits
Challenges	Dimension Control, Noise, Poor drive capability	Low Temps, Difficult to fabricate	Low fanout, Room temp operation, feedback from other devices	Difficult to fabricate, Properties poorly understood	Thermal & Environmental Stability, 2-term devices, Need new archs
Maturity	Demonstrated	Demonstrated	Demonstrated	Demonstrated	Demonstrated

continues until it is replaced by newer technologies with better performance, size, or other characteristics.

Several potential replacements for silicon CMOS are being investigated. Research in most of these devices is in the early stages, and it is not yet possible to manufacture them in large numbers or as reliably as current commercial devices. A common characteristic of these devices is they are more prone to defects than current silicon CMOS. Adoption of these devices may be accelerated through the use of defect tolerance techniques. While not yet mature, these technologies show the potential to overcome some of the limitations of silicon CMOS. This section discusses several of the alternative technologies.

2.3.1 Next Generation Devices. Table 2.4 shows several emerging technologies [HBZB02]. Important characteristics, capabilities, and challenges of these technologies are discussed in this section.

2.3.1.1 Molecular Crossbars. Molecular devices are a promising technology. Molecular logic devices are based upon electron transport through a single molecule [HBZB02, WK00]. Most experiments to date have been with two-terminal

devices, although three terminal devices are now being proposed. The two-terminal devices being studied are typically composed of thousands of molecules in parallel, operating as digital switches or as analog diodes.

Molecular switches are most commonly constructed from a single molecular layer sandwiched between two lithographically patterned metal lines. The molecular material between the crossing metal lines can be one of several different materials, most commonly electrically configurable bistable molecules such as the *rotaxane* and *catenane* families of molecules pursued by Hewlett-Packard and UCLA [CWB⁺99, PJS⁺01]. While the junctions are currently one layer thick in the Z axis, they are roughly 10^4 atoms across in the X and Y directions. In theory, the devices should scale with the width of the metal lines. thus, molecular devices have the potential for very high density.

The molecular switch can be modelled as a diode with a switchable threshold (i.e., turn-on) voltage. The switches are set or reset by electrochemical reduction or oxidation of the sandwiched molecules. Application of a positive programming voltage changes the operating characteristics of the molecular junction, opening or closing the switch. Once programmed, the status of a junction can be read by applying a negative voltage [KW02]. Molecular crossbars can be used for both memory and logic. Simple AND, OR and NOT gates have been fabricated using resistor-diode designs [ZS02, CWB⁺99].

One long term vision for molecular electronics combines both molecular memories and logic devices. Researchers have yet to fabricate large numbers of these devices, but propose to organize molecular switches in crossbar structures [ZS02, LKC04]. A crossbar design is shown in Figure 2.17. The array is created by two orthogonal planes of parallel metal lines. A molecular switch is created at each intersection. Current experiments have created arrays of 4x4 or 8x8 elements [LKC04]. These crossbar arrays have been programmed as multiplexers and demultiplexers as well as a 4x4 memory array.

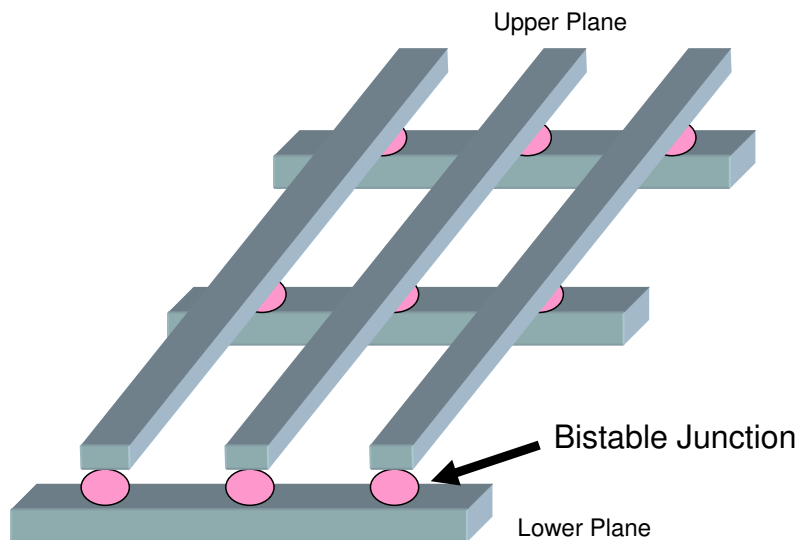


Figure 2.17: The molecular crossbar paradigm consists of perpendicular sets of parallel wires with bistable junctions at each intersection [ZS02]

The interconnect requirements for molecular circuits is extremely demanding. The junctions themselves are very small. Thus, designs using molecular devices must make the most efficient use of interconnect. This makes the use of regular, array-based structures the most likely design, indicating that logic functions will likely be memory-based [HBZB02]. Logic functions will be implemented as pre-calculated truth tables stored in molecular memory and accessed via dense, self-assembled interconnect.

Since the oxidation reactions are reversible, molecular devices will be reconfigurable. When used as programmable logic devices, molecular devices have several advantages over CMOS:

- High density
- Fewer devices needed to store configuration bits
- Fewer wires

In a CMOS FPGA, six or seven transistors are used to store each configuration bit. A molecular device can store the configuration bit in a single junction. In

addition, fewer wires required in a molecular design. In CMOS, two wires are used for address lines to configure a switch, while two more data lines are needed to read it. For molecular devices, only two wires are needed for both functions [KW02]. Fewer wires means less routing, with fewer locations for defects to occur.

Molecular devices, as might be expected, have several limitations to be overcome. One of the problems with these devices is the difficulty in creating inverting devices [ZS02]. While more complex, inverting structures have been demonstrated [KW02], they are larger and more complicated than noninverting devices. Another problem is the lack of signal gain [LMSL05], which limits the ability of a molecular logic gate to drive other devices. Nanoscale wiring is also a challenge, as lithography will not be able to create the thin wires necessary to produce levels of integration greater than CMOS. Alternative methods of creating the lines are under development, including nanoimprint, interference lithography, and self-assembly [LMSL05].

One possible use for molecular devices is in hybrid systems combined with sub-100 nm CMOS [LMSL05]. The problem of low voltage gain can be overcome by using attached MOSFETs as drivers to amplify the output of the molecular switch. The Complementary Molecular (CMOL) architecture overlays a molecular crossbar on top of an underlying standard CMOS device [LMSL05]. Connections between the two levels are made using vias, as illustrated in the top diagram of Figure 2.18). Rotation of the crossbar array relative to the CMOS, shown in the bottom part of the figure, reduces the need for precise alignment and allows for addressing of individual molecular rows and columns with CMOS circuits that are much larger in size [LMSL05]. The CMOL team predicts that this method will allow the creation of device densities in excess of 10^{12}cm^{-2} , at least three orders of magnitude higher than CMOS [LMSL05, page 3.11]. A CMOL FPGA was recently proposed in [SL05].

Chemical self-assembly has been proposed as a low-cost method to fabricate molecular devices [ZS02]. Self-assembly may allow the fabrication of devices with a higher density than silicon CMOS at a much lower cost. The disadvantage is that self-

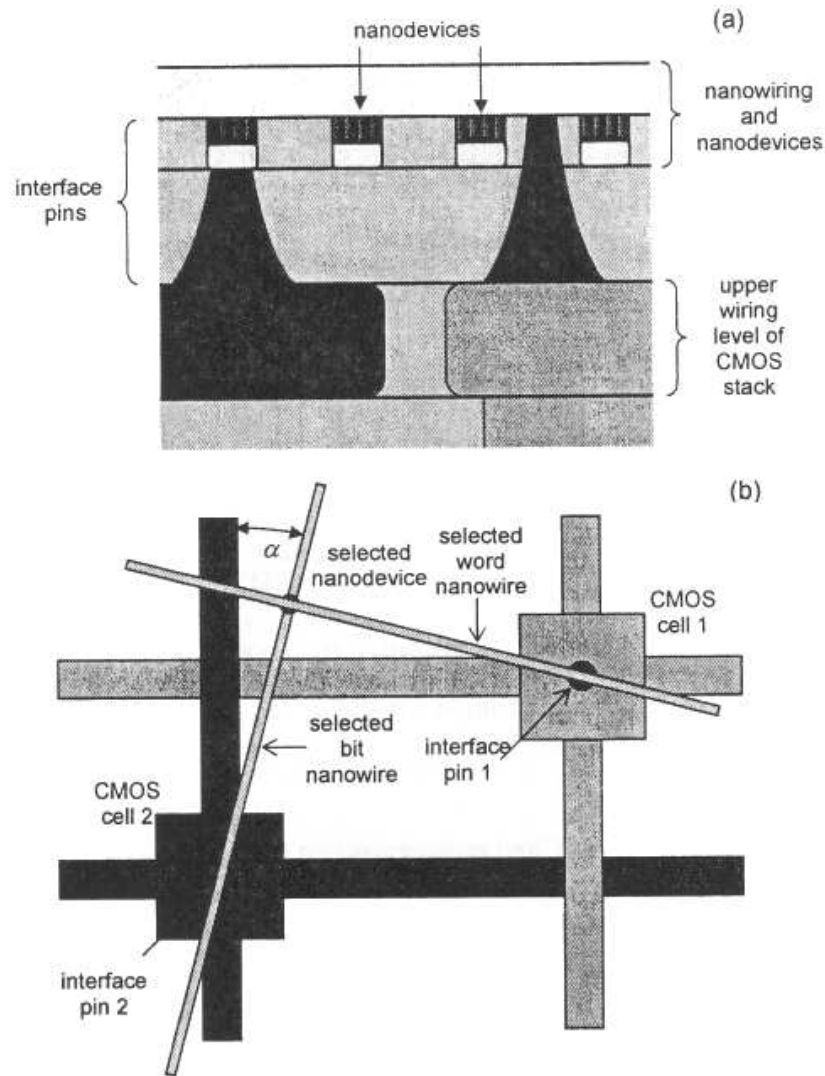


Figure 2.18: A Generic CMOL circuit overlays a molecular crossbar on top of a standard CMOS device [LMSL05]. The top part of the figure shows the vias connecting the molecular crossbar array to the underlying CMOS devices. The bottom part shows the rotation of the crossbar relative to the CMOS layer that allows more more precise connections.

assembly is innately a statistical process, and is subject to defects at a much higher rate than conventional CMOS [ZS02, KW02]. Indeed, it has been proposed that the development of molecular devices and defect-tolerant architectures must proceed in tandem [HBZB02]. To build a functional circuit from a crossbar structure containing many defects, the devices must be tested, defects located, and the final circuit configured to use only the operational devices. The crossbar structure is well suited to this approach, as long as sufficient redundant rows and columns are available. Methods for achieving defect tolerance in this manner will be discussed in Section 2.6. Molecular memories are well suited to the reconfiguration approach, as the physical area of a configuration bit is no larger than the intersection between the two metal lines. In a conventional CMOS FPGA, a configuration bit requires approximately twenty times this area [KW02]. Current FPGAs do not compete well with fixed-design Application Specific Integrated Circuits (ASIC) in terms of layout area required to implement a design. This is because a huge amount of area is required to implement the configuration memory bits. The Teramac reconfigurable architecture devotes approximately 90% of its area to configuration memory [KW02]. Molecular memories reduce this area requirement tremendously. For this reason, the reconfigurable row-column array based architecture may be ideally suited for defect tolerant computing with molecular devices. It is partly for this reason that FPGAs are chosen as a starting architecture for this research. FPGAs will be discussed in detail in Section A.1.2.

2.3.1.2 Single Electron Transistors. Single electron transistors (SETs) are constructed much like larger MOSFETs, but have a channel that is so small that only a single electron can occupy the channel at a time [Ris02, T⁺96]. SETs are three terminal devices like normal FETs, in which current is controlled electrostatically. SETs are fabricated on a substrate of metal or an insulator (i.e., Silicon-On-Insulator (SOI)). The PN junctions are replaced by high-resistivity tunnel junctions. After one electron has entered the channel, its electric charge prevents any other electrons from entering the channel until it has tunnelled out of the channel.

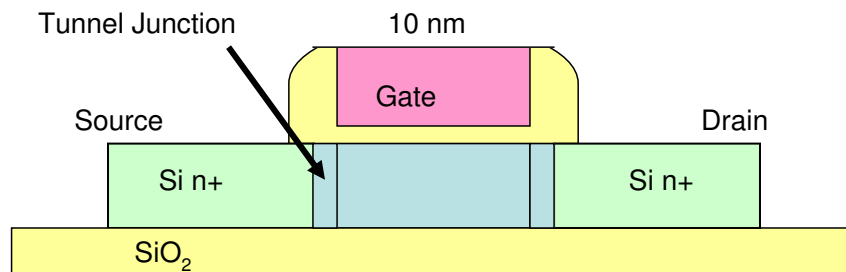


Figure 2.19: A Single Electron Transistor fabricated as SOI with tunnel junctions [Ris02].

Operation of SETs is similar to larger MOSFETs, although there are some differences. Because only one electron is in the channel, these devices are very sensitive to electrical noise and have slower switching speeds [Ris02]. SETs are currently very difficult to fabricate reliably and only small numbers of devices have been produced. In addition, cryogenic cooling is currently required [HBZB02], a significant limitation to widespread use.

2.3.1.3 Carbon Nanotube Devices. Carbon nanotubes have been under heavy research over the last several years. A carbon nanotube is a molecule composed of many atoms of carbon in a hollow cylindrical arrangement. Nanotubes typically have a diameters from 1-20 nm and lengths from 100nm to several microns [HBZB02]. Varying the diameter and the arrangement of the atoms can greatly influence the electrical properties of the nanotube, allowing it to be a conductor, insulator, or semiconductor. When used as a semiconductor, varying the diameter also varies the bandgap of the material. Nanotubes can also be doped with impurities, allowing the creation of PN junctions. Field effect transistors can be constructed from nanotubes [MSS⁺98], and arrays of FETs have been created [CAA01].

The biggest limitation on carbon nanotube devices is control of production of the nanotubes [Bou03]. All the known processes to produce carbon nanotubes produce nanotubes of all sizes and lengths. To make devices, the nanotubes must be separated into groups of similar size and behavior. This process is currently done manually in

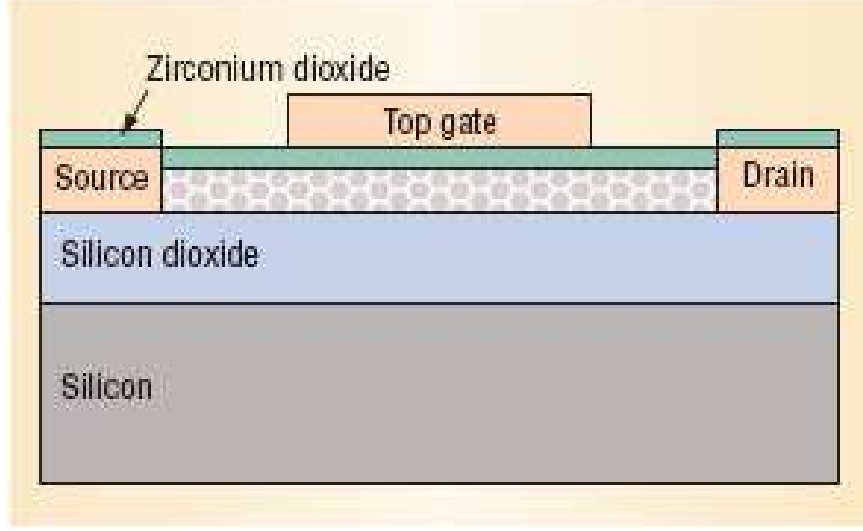


Figure 2.20: A Carbon Nanotube FET fabricated as SOI with tunnel junctions [Bou03].

labs using scanning tunnelling microscopes. For large scale production of transistors, methods must be developed to control the fabrication of the carbon nanotubes.

2.3.1.4 Quantum Cellular Automata. The QCA paradigm is differentiated from CMOS by the locality of interaction in which each cell talks only with its nearest neighbors [HBZB02]. Communication occurs via electromagnetic fields and quantum tunnelling rather than charge flow in the conductor. Custom boolean gates have been designed using QCA [AOT⁺99]. A QCA is an array of quantum dot cells. Each cell is comprised of four dots located at the vertices of a square, as shown in Figure 2.21. When the cell is charged with two excess electrons, they occupy diagonal sites as a result of mutual electrostatic repulsion. The two states are used to represent logic ‘0’ and ‘1’, shown in the left and right sides of the figure, respectively. A polarization change in a QCA cell is induced by causing an electron to switch positions in one of the dots by applying an electric field.

The fundamental QCA logic device is the three input majority gate, shown in Figure 2.22. The right side of the figure shows the voltages at the edges of the central cell with the inputs $A = B = C = 1$. The output is read by applying a voltmeter at

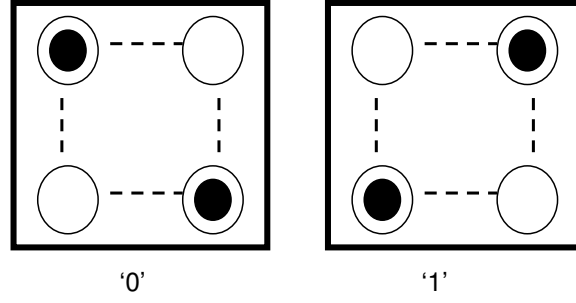


Figure 2.21: Quantum Cellular Automata. Two possible states for the electrons in the Quantum Cell [AOT⁺99].

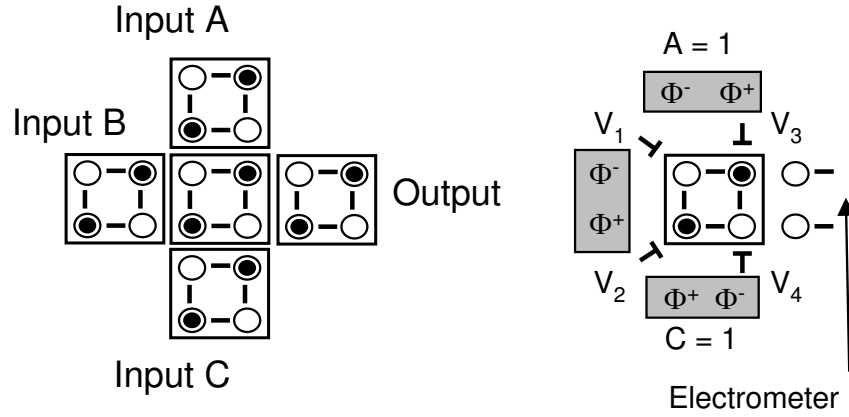


Figure 2.22: QCA Majority Gate. The right diagram shows the electric fields applied to induce the proper input states. [AOT⁺99].

the rightmost cell. The majority gate can also be used to form boolean logic gates. It can be programmed to act as an AND or OR gate by fixing one of the three inputs.

QCA devices have the potential for very high density. Interconnect is created with lines of QCA cells. QCA designs for a memory cell and a full adder are shown in Figure 2.23. Unfortunately, current QCA devices require cryogenic temperatures and have small drive capabilities, limiting fanout. Designs utilizing feedback require further investigation.

Additional information on QCA is provided in Section 10.3.

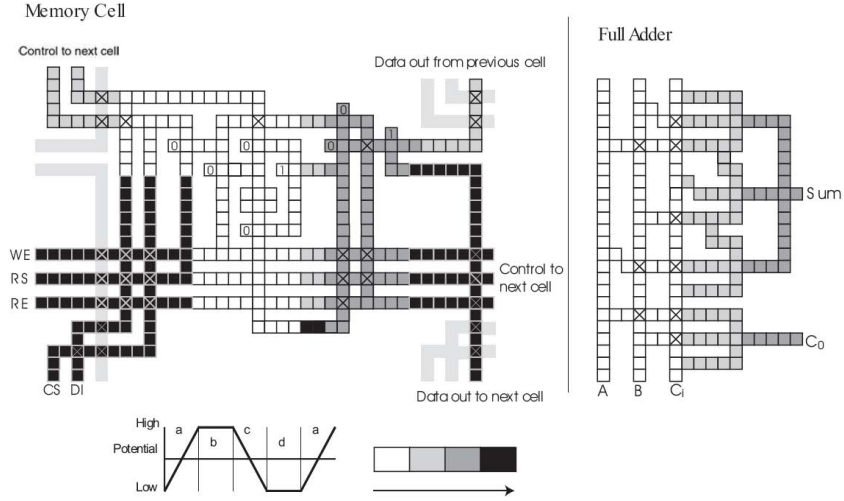


Figure 2.23: QCA Memory Cell and Adder Designs [FNS01].

2.3.1.5 DNA Self-Assemble Devices. A potential solution to the lithography problem is the use of self-assembly. DNA-guided self-assembled devices have recently been proposed [DVP⁺04]. A computer architecture using these types of devices was proposed in [DPTV04]. This technique takes advantage of the ability of DNA to form pairs of complementary strands. Proper selection of DNA elements guides the placement of semiconducting, conducting, and insulating rods to form three dimensional structures that implement logical devices or look-up tables.

The DNA assembly process is by nature difficult to control precisely, and is a statistical process. The proposed design paradigm is once again “build a bunch of devices, then discover which ones work.” Defect rates for these types of devices is currently around 2% [DVP⁺04, page 1216]. Proposed uses for these devices include a form of Content Addressable Memory (CAM). A simple adder could be constructed by creating large number of DNA devices, each of which is coded to respond to a certain input arguments (a *query*). This type of design is called an *oracle*. The devices randomly assemble, each implementing an addition operation of two arbitrary inputs. The oracle operates by querying the devices with the problem inputs (i.e., “What is 3+5?”). Only the DNA-assembled devices which were constructed to implement

those two operands would respond by raising a flag signal. The oracle would then query the device raising the flag to determine the answer. Of course, it is possible that none of the randomly assembled addition devices implement a particular pair of inputs, so large numbers of devices may be required to provide responses to all input combinations with an acceptable probability.

2.3.1.6 Other Potentials. The feasibility of Rapid Single Flux Quantum logic has already been demonstrated. RSFQ is dynamic logic based on a superconducting quantum effect in which the storage and transmission of flux quanta defines device operation. The RSFQ device is a superconducting ring that contains a Josephson junction plus an external resistive shunt [HBZB02, page 34]. The storage element is the superconducting ring, while the switching element is the Josephson junction. RSQF logic uses the presence or absence of the flux quanta in the superconducting loop to represent a logical state. The circuit operates by closing the Josephson junction, ejecting the stored flux quanta. This generates a quantized voltage pulse that can be used to trigger other RSQF devices.

RSFQ has the potential to be extremely fast, with predicted speeds from 100-750GHz [HBZB02]. RSFQ also has several disadvantages. RSFQ currently requires the use of low temperature superconducting Josephson junctions ($5K$). In addition, it is predicted that size may be a limiting factor. Operation is limited by the maximum magnetic penetration depth in the device, which depends on size. One estimate for the minimum size for RSFQ devices is 100 nm for low temperature superconductors, and 500nm for high temperature superconductors [Com00].

2.3.2 Common Features of Developmental Devices. It is impossible to predict whether any of these technologies will be widely adopted. The physics of these devices must be better understood, and solutions must be found to the limitations of the current prototype devices. To compete with current silicon CMOS, any potential replacement must compete favorably in device density, speed, and power consumption. It must also be economically feasible to justify the cost of shifting production

from CMOS to a new technology. It is likely that these technologies will first be used for special purpose applications, perhaps in combination with conventional CMOS.

2.4 Fault Tolerance

This section provides a general review of fault tolerance techniques. Real systems are built from components that are subject to failure. Components that functioned when the system was new can degrade and fail in operation. In addition, given the complexity of digital systems, it is possible that the *design* of the system is flawed and does not produce correct results in all situations. The system designer has two choices: attempt to design a system that cannot fail, or design a system that tolerates failures. As discussed in the previous sections, tremendous effort has been expended in the past to design systems that do not fail. But as systems become more complex, it is increasingly difficult to guarantee correctness. For this reason, fault tolerance enables the use of systems that otherwise would have failed.

This section is organized in three parts: Section 2.4.1 introduces key fault tolerance definitions. Section 2.4.2 discusses the major strategies and concepts used in fault tolerance. Finally, Section 2.4.3 discusses hardware-based techniques for fault tolerance. These techniques form the foundation of this research.

2.4.1 Faults and Testing Concepts. This section introduces the key definitions in fault tolerance. Fault types provide an understanding of what kinds of failures occur in digital circuits. The device testing process is introduced, with emphasis on the difficulty in testing large circuits. Finally, methods for characterizing the reliability of the overall system are examined.

2.4.1.1 Key Definitions.

Error is a manifestation of a fault in the system, in which the current logical state differs from the correct state.

Fault is an anomalous physical condition. Causes include design errors, manufacturing defect, device deterioration, electrical noise, or other environmental factors.

Failure denotes an inability to perform a desired function because of *errors* in the system caused by one or more *faults*. A failure can mean an incorrect result, no result at all, or the violation of a design parameter (e.g., a correct result returned too late to be of use).

Fault Tolerance is the ability of a system to perform an intended function in the presence of errors caused by one or more faults. These faults may be temporary (e.g., caused by electrical noise) or permanent (e.g., device degradation), and may occur at any time during system manufacture or operation.

Defect Tolerance is a subset of fault tolerance, limited to faults that were present in the system at its initial fabrication or assembly. Defect tolerance is the ability of a system to perform an intended function in the presence of errors caused by one or more manufacturing faults.

2.4.1.2 Fault Types. Design of a fault tolerant system requires understanding of the ways the system can fail. It is necessary, therefore, to characterize *fault types* that can occur and estimate their impact, severity, and probability of occurrence. Then a fault tolerance technique can be chosen to detect, isolate, and recover from the fault.

Faults can be characterized temporally:

- Transient faults are temporary and non-recurring. Errors observed due to transient faults are called *soft errors*. The rate at which soft errors occur in operation is called the *Soft Error Rate* (SER).
- Intermittent faults are temporary, but recur in operation.
- Permanent faults are also called *hard faults*. Errors due to permanent faults are usually repeatable and are called *hard errors*. A manufacturing defect can be considered a hard fault, although in some cases its effects are intermittent.

Depending on the fault type, its effects can be observed and characterized at the digital or logical level, as well as at the circuit level. Not all defects result in faults; likewise, not all faults result in errors or failure [Raj92]. The goal of testing is to determine which faults cause failures in the system, and then test for those faults. Fault effects can be classified in many ways. Two common classifications are *logical faults* and *parametric faults*. Logical faults produce errors in the logical or Boolean state of the system. Three common logical faults are *Stuck At* faults, *von Neumann* faults, and *Bridging* faults. Stuck At faults occur when the input or output of a gate is literally stuck at a particular logic value (i.e., ‘1’ or ‘0’). A von Neumann fault occurs when a input or output is inverted from the correct value [vN56]. Bridging faults commonly occur between adjacent signal lines, and cause the value of one of the lines to match the value of the other.

Faults can also be specific to a key aspect of the circuit architecture. For example, FPGA architectures have faults not found in other devices. Similar to bridging faults, *programmable interconnect open and short* faults occur in connections between interconnect lines in the FPGA. These connections can be stuck open or closed, and can affect the operation of the FPGA in unusual ways thereby changing the operation of the application circuit [RN95].

Parametric faults change the performance of the system, but may not induce logical errors. An example is the *timing fault* which is sometimes caused when defects in a transistor cause it to switch slower than designed. A timing fault may manifest itself as an error if the result of a circuit is not available until after the clock edge latches an incorrect result. *Current faults* occur when the circuit draws too much current, due either to leakage or a short circuit.

2.4.1.3 VLSI Test. Testing of VLSI circuits for correct function is a difficult and time consuming process. For simple circuits (e.g., an 8 bit adder), it is possible to perform *exhaustive testing* where every possible input is applied and the correctness of the output is verified. Similarly, state machines can be forced into every

state and all transitions verified. For more complex circuits, it is infeasible to apply every possible input vector or force the state machines into every state. In this case a subset of all the possible test patterns is used. Careful selection of the test patterns provides the maximum possible *test coverage* with the fewest test vectors.

Faults can occur at any node within a device. It is normally only possible to observe the circuit through the output pads, although sometimes special test pads are included in the layout. Even so, the number of points at which circuit operation can be observed is small compared to the number of inaccessible internal nodes. A circuit path is *sensitized* to a fault if a set of inputs can propagate the fault effect to an observable output. The general approach to test pattern generation considers each possible fault and how its impact can be propagated to an observable output. Input test vectors are chosen such that a fault can be detected on the output pins.

The number of test vectors can rapidly become extremely large. Fortunately, many faults produce the same incorrect output when a particular test vector is applied. These faults are called *equivalent*, and one test can detect all of the faults in each equivalent set. In this manner, the total number of test vectors is reduced, while still maintaining fault coverage.

To test more complex circuits, designers can add additional circuitry to support testing. The most common technique is *boundary scan* testing. The industry standard for boundary scan is IEEE standard 1149.1, also known as JTAG (for Joint Test Action Group). The basic idea of a boundary scan is to introduce a memory scan cell at each input and output (I/O) pin [Raj92]. The boundary scan cells are interconnected to form a chain. The chain is connected to an input pin and output pin which allows test vectors to be shifted serially onto the chip and test results to be shifted off. In addition to I/O cells, the circuit can connect to internal storage elements as well. These cells are transparent to normal operation of the circuit. During testing, however, the scan chain can load test vectors into the device and capture the results so that they can

be shifted out. In this manner, it is possible to access and test internal nodes in the chip using a small number of test pins.

The JTAG pins are sometimes used for other purposes. Field Programmable Gate Arrays (FPGAs) often use the JTAG pins to shift configuration data into the chip. FPGAs are discussed in more detail in Section A.1.2.

Microchips are typically tested at fabrication by special automated test equipment. It is also possible to embed test circuitry on the chip itself to allow the chip to generate its own test patterns and report faults. This technique is called *Built-In Self Test* (BIST).

2.4.1.4 Reliability Measurements. The impact of defects and other faults on system performance can be quantified. First, the *yield* of the manufacturing process is defined as the percentage of produced chips that function correctly. Modern silicon CMOS processes have yields in the range from 60-90%. Since as many as 40% of production chips contain one or more faults, the effectiveness of the test process is key in finding defective chips before they are shipped. *Test coverage* is the fraction of faults that are detected by applied test vectors. While time consuming, application of more test vectors results in a higher test coverage. This can greatly decrease the probability of defective chips being falsely accepted. The probability, $P(\alpha)$, that a defective chip is accepted is

$$P(\alpha) = 1 - Y^{1-\eta} \quad (2.9)$$

where η is the test coverage, and Y is the yield of the process.

Suppose a given manufacturing process has 90% yield [Raj92]. The chips are subject to testing that provides 80% fault coverage (i.e., the tests find 80% of the possible faults). Therefore, roughly 2% of the chips are defective and not detected by the test process. Clearly, it is desirable to provide the highest possible test coverage to minimize the number of defective chips which are shipped to the customer. In this simple example, fault tolerance was not considered. It will be shown later that fault

tolerance techniques can increase the yield by correcting errors that would otherwise cause the chip to fail.

At the system level, several reliability concepts can now be defined.

Reliability, $R(t)$, is the conditional probability a system can perform its designed function at time t , given it was operational at $t = 0$.

Point Availability, $A(t)$, is the probability that a system can perform its designed function at time t . Availability is sometimes expressed as a steady-state value, either as a probability the system functions correctly at any given instant, or as an amount of down time over an interval (e.g., two minutes of down time per year).

Mean-Time-To/Between-Failures (MTTF/MTBF). MTTF/MTBF is the expected time until system failure.

Mean-Time-To-Repair (MTTR). MTTR is the expectation of how long it takes to return a failed system to operation.

Coverage. The coverage of a fault tolerance technique is the probability a circuit is functional given a particular fault occurs (i.e, $P(Functional|Fault)$).

Functional. The circuit performs its design function without errors.

For example, suppose that components fail at a constant rate. Reliability, then, can be expressed as

$$R(t) = e^{-\lambda t}, \quad (2.10)$$

where λ is the sum of the failure rates of the components [AL81]. Mean time between failures is simply

$$MTBF = \int_0^{\infty} R(t) dt \quad (2.11)$$

When fault tolerance techniques are incorporated, the reliability of the system is [Nel90]

$$R_{system} = P(\bar{E}) + P(F|E)P(E), \quad (2.12)$$

where E is the fault event, \bar{E} is the no-fault event, and F is the event that the circuit functions correctly.

A related IC failure rate metric is the *failure unit* (FIT) defined as one failure in 10^9 device hours [Raj92].

2.4.2 Fault Tolerance Strategies. All fault tolerance techniques incorporate *redundancy*, or hardware or computations not normally required for system function. Redundancy can be either spatial or temporal. The general goal of fault tolerance is to increase the probability of correct function while minimizing some cost function. Design factors considered may include performance, production cost, design complexity, circuit area, power consumption, and test complexity.

Fault Tolerance strategies include one or more stages [Nel90]:

- Error Detection.
- Fault Masking. Dynamic correction of generated errors.
- Fault Confinement/Containment. Prevention of error propagation across defined boundaries.
- Fault Diagnosis. Identification of the faulty module responsible for a detected error.
- System Reconfiguration and Repair. Elimination or replacement of a faulty component, or a mechanism to bypass it.
- System Recovery. Correction of the overall system to a state acceptable for continued operation.

These stages will be discussed in the following sections.

2.4.2.1 Error Detection. Error detection is done either online, while the system is in normal operation, or offline. In offline detection, normal operation is suspended at specified intervals while error detection is performed. Error detection in memory is done easily most commonly using parity bits . Error detection in logic is more difficult, using input encoding, shifting, and other methods. Coding theory can be used for data transmission on the chips (e.g., Reed-Solomon codes, convolution codes, etc. [Skl01]).

Common checking techniques for error detection are:

Replication Checks. The system performs an operation more than one time and compares the results.

Timing Checks. Raises an exception when the operation takes longer than a specified limit to complete. This method is well suited to detect parametric errors.

Reversal Checks. From the output, the system attempts to determine the inputs to the operation. This method is appropriate for reversible operations (e.g., check a square root function by squaring the output).

Coding Checks. Based on redundancy in the representation of the object. This method is often used to detect memory errors (e.g., parity checks, etc.)

Reasonableness Checks. Compares the output against a set of expected values. This method is also called a range check (e.g., for a computation producing a direction, the result should be $[0 - 359]$ degrees). Reasonableness checks are more commonly used in the application or in the operating system. For example, the operating system commonly checks the ranges of memory address requests to prevent an application from accessing memory outside its assigned memory space.

Structural Checks. Performed on data structures at the application or operating system level.

Diagnostic Checks. Offline testing performed by stimulating the system with specific test vectors designed to detect faults.

2.4.2.2 Fault Masking. A system that uses fault masking produces even in the presence of errors. In this way, faults are masked from observers outside the system. Examples include Triple-Modular Redundancy (TMR), Majority Voting, and von Neumann Multiplexing. These techniques are discussed in Section 2.4.3.

2.4.2.3 Fault Diagnosis. Once an error is detected, fault diagnosis can be used to isolate the cause of the fault to a particular module or device. Fault diagnosis is a necessary step to the application of another stage, *System Reconfiguration and Repair*.

2.4.2.4 Fault Confinement/Containment. Fault Confinement/Containment limits the impact of a fault to a small area or portion of the system. The goal is to limit the damage a fault does to the system operation and to minimize the recomputation to recover from the fault. A typical method is to break the operation of the system into *atomic operations* [AL81]. If a module detects a fault, the entire atomic operation is recomputed.

In the system architecture, containment boundaries are established in two ways: each module checks its own outputs, or each module checks its own inputs. Care must be taken to cover faults occurring at the interfaces themselves.

2.4.2.5 System Reconfiguration and Repair. System Reconfiguration and Repair changes the structure of the system to recover from detected faults. One technique uses redundant modules. When a module fails, the system disables the faulty module and activates a replacement module. In reconfigurable computing systems based on programmable logic devices (PLDs), the application logic on the PLD is reconfigured and re-routed to avoid the faulty portions of the PLD.

Replacement units can be either ‘hot’ or ‘cold’. Hot spares operate concurrently with the rest of the circuit prior to the fault and need no initialization when a failover occurs. A cold spare is either not powered, or is used for other tasks, and requires initialization when it is put into use. Initialization time is a key factor in performance for cold spares.

2.4.2.6 System Recovery. System Recovery returns the system to a previous correct state or to a *recover point*. Processors, for example, can be rolled back to a previous instruction and register state.

Backward error recovery restores the system state to a previous known, error-free state. This method is effective when recovering from unknown errors. Since computations must be recalculated, there is sometimes a significant performance penalty [AL81]. Furthermore, it may not be possible to recover to an error-free state. An example of backward error recovery is Single Instruction Retry (SIR). Used at the processor level, SIR re-executes an instruction that produces an error. SIR is sometimes used at the processor level to overcome soft errors.

Forward error recovery corrects an error without recalculation. For this to work, the failure mechanism must be well understood and predictable. Methods to correct each fault type are then developed. Forward error recovery, then, is more difficult and less effective at handling unknown errors.

2.4.3 Hardware Techniques for Fault Tolerance. This section introduces several fault tolerance techniques that provide fault tolerance at the hardware level. They are used at the individual gate level, at the module level, and higher. For several of these techniques effectiveness models have been developed. These models are important as they quantify the effectiveness of proposed techniques and designs. The techniques introduced in this section are *R-Modular Redundancy*, *Cascaded Tri-Modular Redundancy*, *NAND Multiplexing*, *Duplication With Comparison*, *Concurrent Error Detection*, and *Reconfiguration*.

2.4.3.1 R-Modular Redundancy. R-modular Redundancy (RMR) implements R units working in parallel and compares the outputs with a majority gate. When $R = 3$, the technique is called *Tri-Modular Redundancy* (TMR). A RMR module behaves identically to the original module, but has a higher probability of producing the correct output. The hardware cost for this improvement is $Rn + M$, where n is the number of devices in the original module, and M is the number of devices needed to construct the majority gate.

In a performance model of RMR, a chip is composed of N_{total} devices organized into modules, each containing N_c devices [FNS01]. Each device fails with a probability p_f . The probability the chip fails is minimized under the condition $N_cp_f \ll 1$. Each module functions correctly if every device in it is functional, or when

$$P_{module,works} = (1 - p_f)^{N_c} \approx e^{-N_cp_f} \text{ (for } p_f \ll 1\text{)}. \quad (2.13)$$

The probability the module fails when $N_cp_f \ll 1$, is

$$P_{module,fails} = (1 - P_{module,works}) = N_cp_f. \quad (2.14)$$

A group consisting of R modules and a majority gate functions correctly when at least $(R + 1)/2$ modules are functional. The probability that the $mmods = (R + 1)/2$ modules are functional is $P_{mmods,works}$. The probability that the majority gate functions correctly is

$$P_{majgate,works} \approx e^{-mBp_f} \quad (2.15)$$

where B is the number of outputs of each of the R modules and the majority gate. The majority gate is composed of mB devices, where m is some constant.

Assuming the majority gate and R modules fail independently, the probability that the group of modules fails is thus

$$P_{group,fails} = 1 - P_{mmods,works}P_{majgate,works}. \quad (2.16)$$

This equation can be expanded and simplified as

$$\begin{aligned}
P_{group,fails} &\approx 1 - P_{mmods,works}(1 - mBp_f) \\
&\approx 1 - P_{mmods,works} + P_{mmods,works}mBp_f \\
&\approx P_{mmods,fails} + mBp_f
\end{aligned} \tag{2.17}$$

The probability that a majority of R modules fails is

$$P_{mmods,f} = \binom{R}{(R-1)/2} P^{(R-1)/2} Q^{(R+1)/2} + \dots + \binom{R}{1} P Q^{(R-1)} + \binom{R}{0} Q^R \tag{2.18}$$

where $P \equiv P_{module,fails}$ and $Q = 1 - P$. When $Q \ll 1$, (2.18) reduces to the first term, and by using (2.14), (2.17) becomes

$$P_{group,fails} \approx C (N_c p_f)^{(R+1)/2} + mBp_f, \text{ where } C = \binom{R}{(R-1)/2}. \tag{2.19}$$

The number of devices in a group is $RN_c + mB$, so the total number of groups is $N_{groups} = N_{total}/(RN_c + mB)$. The probability that the whole chip with N_{total} devices fails (when $P_{group,fails} \ll 1$) is approximately

$$P_{chip,fails} \approx N_{groups} \times P_{group,fails} = \frac{N_{total}}{RN_c + mB} \left[C (N_c p_f)^{(R+1)/2} + mBp_f \right]. \tag{2.20}$$

Solving for $dP_{chip,fails}/dN_c = 0$ gives the optimum module size (N_c) for a given p_f , which when substituted into (2.20), yields the minimum failure probability.

RMR can be quite effective. Figure 2.24 shows the maximum tolerable individual device defect rate versus amounts of redundancy (i.e., R) for a test design [FNS01]. In this figure, $\varepsilon = 0.1$ is the maximum acceptable failure probability for the chip (i.e., $\varepsilon = 1 - Yield$). B is the number of outputs from each of the R modules and the majority gate. The device overhead to implement the majority gate is $m = 20$. For

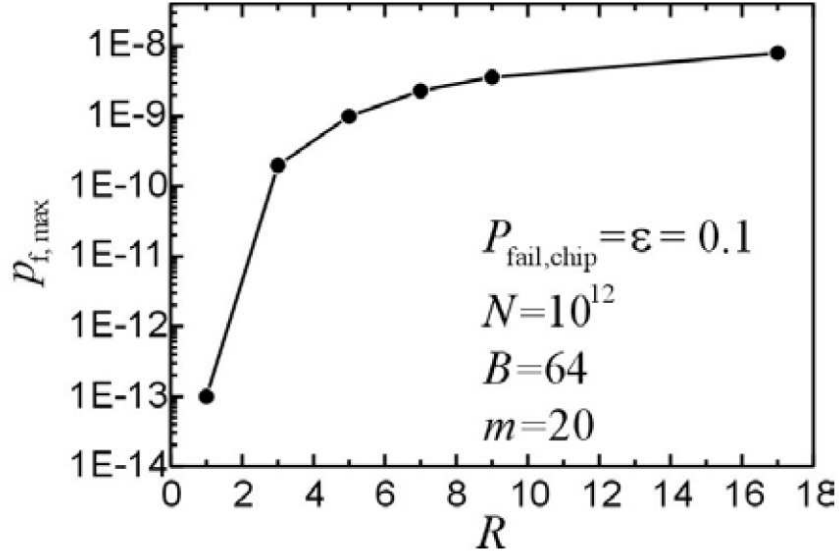


Figure 2.24: RMR Performance (Max allowable p_f versus Redundancy, R) [FNS01].

a design with no redundancy (i.e., $R = 1$), the design meets yield requirements only if the individual defect rate is $p_f \approx 10^{-13}$. Incorporating RMR with $R = 5$ raises the allowable defect rate to $p_f \approx 10^{-9}$. Thus, the same probability of chip reliability can be achieved with devices four orders of magnitude less reliable

2.4.3.2 Cascaded Tri-Modular Redundancy. RMR benefits can be increased by connecting RMR or TMR modules in series. The outputs of three TMR modules can be connected to a majority gate to make a second order TMR module, or *Cascaded TMR* (CTMR) module. The probability of correct output with a CTMR module of the i th order is $P_w^{(i)}$, or

$$P_w^{(i)} = (1 - p_f)^{mB} \left[(P_w^{(i-1)})^3 + 3 (P_w^{(i-1)})^2 (1 - P_w^{(i-1)}) \right] \quad (2.21)$$

where mB is the number of devices in the majority gate [FNS01]. However, there is no advantage to using CTMR for units containing small numbers of devices, although improvement is possible for CTMR units with large values of N_c . Three improvement regions are identified [FNS01].

1. $N_cp_f > \ln 2$, redundancy affords no advantage.
2. $10^{-3} \lesssim N_cp_f < \ln 2$, redundancy is most effective.
3. $N_cp_f < 10^{-3}$, only first order redundancy offers an advantage.

In region (2), effectiveness scales exponentially with the order of CTMR used. The failure probability is

$$P_{fail}^{(i)} \propto (N_cp_f)^{2i}. \quad (2.22)$$

For region (3), effectiveness scales in accordance with the ratio mB/N_c . The failure probability for this region is

$$P_f^{(i)} \approx \begin{cases} N_cp_f & , \text{ for } i = 0 \\ \frac{mB}{N_{total}} N_cp_f = \frac{mB}{N_c} P_f^{(0)} & , \text{ for } i = 1, 2, \dots \end{cases} \quad (2.23)$$

2.4.3.3 NAND Multiplexing. NAND Multiplexing was originally proposed by John von Neumann in 1956 [vN56]. In early computers, logical functions were realized using vacuum tubes. These devices were prone to failure, and the mean time before failure of a vacuum tube computer was quite low. von Neumann showed that when the probability of gate failure is sufficiently small, and errors are independent, a high probability of a correct result can be achieved using NAND multiplexing.

A NAND Multiplexor reliably performs the boolean NAND operation in the presence of errors that change the operation of the device. A ‘von Neumann fault’ [vN56] inverts the correct output of a NAND gate. The NAND multiplexor circuit performs the NAND operation redundantly, as shown in Figure 2.25, increasing the probability of correct output over a single NAND gate.

Logic signals in the multiplexing technique are implemented by bundles of signals. For example, a NAND gate may have two inputs, X and Y , and one output, Z . Each signal is implemented as a bundle of N signals. If there are no errors in a signal, all N lines in the bundle have the same value. If errors are present, some fraction of

the lines have the opposite value. A threshold, $\Delta \in (0, 0.5)$ is defined such that when no more than ΔN of the lines in the bundle are stimulated (i.e., logic ‘true’ or ‘1’), the logical value of the variable represented by the bundle is interpreted to be ‘false’ or ‘0’. Likewise, at least $(1 - \Delta)N$ lines must be asserted for the logic value of the variable represented to be considered ‘true’ or ‘1’. If the number of asserted lines in the bundle is between these two thresholds $(\Delta N, (1 - \Delta)N)$, the state is undecided, and a malfunction is declared.

The NAND Multiplexor is composed of two parts: the *Executive Stage* and one or more *Restorative Stages*. Each restorative stage is nothing more than two executive stages in series. In most cases, adding more restorative stages or increasing the bundle size N makes the NAND operation more reliable.

The *Executive Stage* contains two parts: a row of N NAND gates in parallel, and a *Permutation Unit* (i.e., block ‘U’). The initial input signals X and Y are implemented as two bundles of N signals. The output of the NAND operation is the bundle Z , which also contains N signals. Prior to the introduction of any errors, all of the signals in each bundle should match the “correct” values (i.e., $X_i = X_j \forall i, j$ and $Y_i = Y_j \forall i, j$). If errors have occurred, some fraction of these lines will contain the logical inverse of the correct value. Without loss of generality, logical true, ‘1’, is defined to be the “correct” value for X and Y , and thus ‘0’ is the correct output Z . Let (X, Y, Z) have $(kx_0 = \bar{x}N, ky_0 = \bar{y}N, kz_0 = \bar{z}N)$ stimulated signals. Thus, the three-tuple $(\bar{x}, \bar{y}, \bar{z})$ is the probability each variable is stimulated, while kx_0, ky_0, kz_0 represent the number of stimulated lines in each respective bundle for stage 0.

In the *permutation unit*, U , the X and Y bundles are randomly permuted and combined into N $X_i Y_j$ pairs. For example, if $N = 4$, one possible permutation is $X_2 Y_3, X_0 Y_1, X_3 Y_0, X_1 Y_2$. These XY pairs are the inputs to the N NAND gates. For a von Neumann error, each NAND gate is subject to an error which inverts the correct logical output with probability ε .

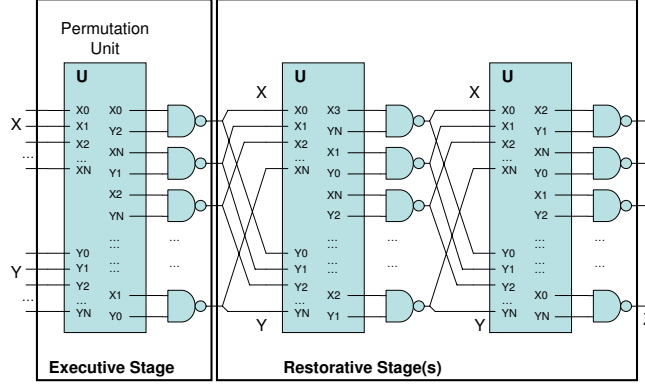


Figure 2.25: NAND Multiplexer

A model of NAND multiplexing has been developed [vN56, HJ02, NPK04] which determines the distribution of the stochastic variable \bar{z} in terms of given \bar{x} and \bar{y} . von Neumann [vN56] determined for large N , the output probability \bar{z} is a stochastic variable with an approximately normal distribution. The upper bound of the probability of gate failure that can be tolerated is $\varepsilon_{max} \approx 0.0107$. The tolerable threshold probability is actually $\varepsilon_{max} = (3 - \sqrt{7})/4 \approx 0.08856$ [EP98]. Beyond this (i.e., $\varepsilon > \varepsilon_{max}$), the failure probability of the NAND multiplexor system is larger than some fixed, positive lower bound, regardless of the bundle size N . Furthermore, for small N , the number of stimulated outputs of the executive stage is theoretically a binomial distribution, although is disputed [NPK04], due to a lack of independence between lines in the output bundle.

A model of NAND multiplexing performance that incorporates the dependence between lines in the output bundle has been developed. The results of this research are found in Chapter VI.

Several system architectures use NAND multiplexing [SNF04, HJ03]. By combining fault masking from NAND multiplexing with fault recovery from reconfiguration, a processor can tolerate defect probabilities as high as 10^{-2} [HJ03]. A processor can be made 90% reliable over ten years of operation, with a defect rate of 10^{-4} and a redundancy of only $R = 50$ [SNF04]. Due to the high levels of redundancy required,

NAND multiplexing has been of limited use. As process sizes get smaller and more devices become available, the technique may see wider use.

2.4.3.4 Duplication With Comparison. Duplication With Comparison [dLKNH⁺04] detects faults using redundant modules. Much like a majority gate, this approach replicates hardware modules. Unlike majority voting, DWC does not attempt to correct an error, but rather detects its presence signals an error handler to the potential fault.

A simple DWC system compares the outputs of two modules. If the outputs differ, an error flag is raised. DWC is of limited utility when used alone, since it is not possible to identify which of the two modules is in error. To overcome this limitation, DWC is sometimes combined with another technique, *Concurrent Error Detection* [LCR03].

2.4.3.5 Concurrent Error Detection. Concurrent Error Detection(CED) detects a fault without stopping circuit operation. While DWC detects faults in the system, CED detects which blocks are fault free [dLKNH⁺04, LCR03]. Figure 2.26 shows the basic concept for a combined DWC/CED scheme. The circuit data outputs are denoted out_1 and out_0 . Three flag signals are used, one to signify a fault has been detected (i.e., the outputs do not match), and two flags to denote the operational state of each of the two logic modules.

After the DWC comparator detects an error, CED determines which module produced the fault. The most common methods are *bitwise inversion*, *recomputing with shifted operands*(RESO), and *recomputing with swapped operands*(REWSO) [dLKNH⁺04, LCR03].

A detailed diagram of the RESO concept is shown in Figure 2.27 [dLKNH⁺04, LCR03]. During normal operation, the A and B inputs are passed through multiplexers to the combinational logic in dr_0 and dr_1 . A clock signal stores the results of the two modules for later comparison. If the two outputs are equivalent, both

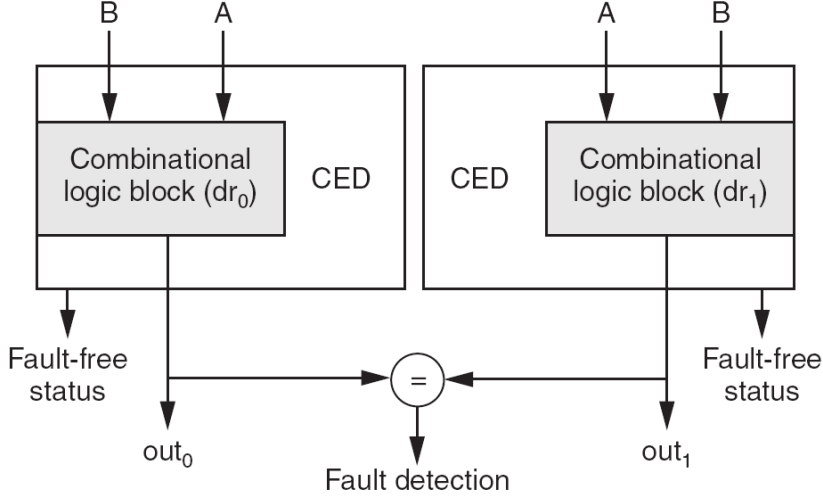


Figure 2.26: Combined Duplication With Comparison (DWC)/Concurrent Error Detection(CED) fault detection scheme [dLKNH⁺04].

output flags *enable_dr₀* and *enable_dr₁* are raised and both outputs are used by later modules. During this second cycle, the operands are shifted prior to use so errors from permanent faults in the combinational logic are different from those obtained in the original calculations. Comparing the results can be used to identify which module is in error. The final *Enable_dr* output flags is then set to tell later modules which input to use.

The benefit of this approach is error detection and isolation is done in only one additional clock cycle. However, a fault occurring in the encoding, decoding, or voter logic produces false positives even when both combinational modules are functional. This problem can be overcome through the combination of DWC/CED modules in larger fault tolerant modules using TMR or other techniques.

2.4.3.6 Reconfiguration. Reconfiguration also can achieve fault tolerance in hardware. Reconfiguration relies on the ability of hardware to modify its configuration to implement different logical structures. It is most commonly implemented in programmable logic devices such as Field Programmable Gate Arrays (FPGAs). The FPGA implement application circuits by programming RAM-based

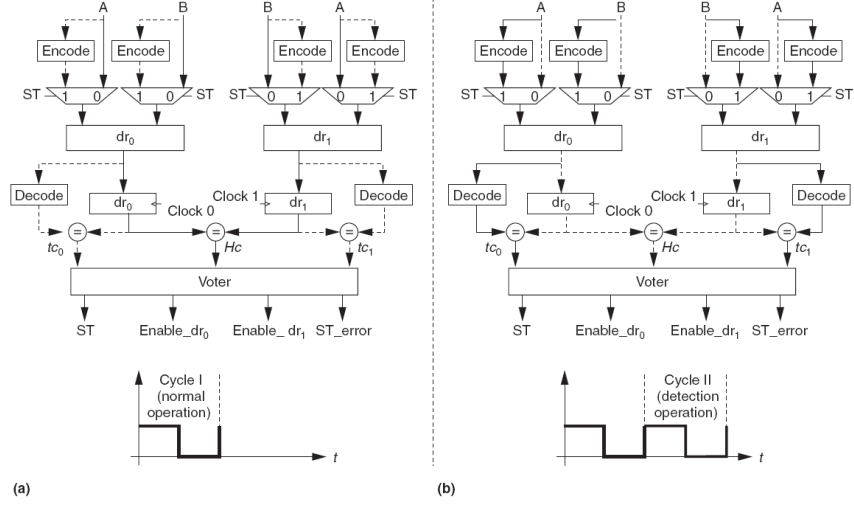


Figure 2.27: Combined Duplication With Comparison (DWC)/Concurrent Error Detection (CED) fault detection operation. Data propagation during normal operation is shown on the left, and in fault detection mode on the right [dLKNH⁺04].

Configurable Logic Blocks (CLB) and setting interconnect switches to connect them. The structure of FPGAs is discussed in more detail in Appendix A.1.2.

Application circuits implemented in FPGAs typically provide better performance than those implemented in software on general purpose processors (GPP). The study of these architectures is often referred to as *Reconfigurable Computing* and is discussed in Appendix A.2.

Reconfiguration is also used to provide fault tolerance. Reconfigurable devices have a large number of configurable logic blocks (CLBs). Provided that the application design does not use all available configurable resources (i.e., CLBs and routing), unused resources can be used to provide hardware redundancy. If portions of the hardware implementing the application circuit fail, the FPGA can be reconfigured to move the affected portions of the application circuit to the spare resources. Reconfiguration overcome manufacturing defects was demonstrated in the Teramac system constructed by Hewlett-Packard [HKS^W98]. Teramac was implemented using 864 inexpensive, low-quality, FPGAs. Upon system configuration, each FPGA was tested

and defects located. A detailed map of roughly 220,000 defects was obtained, and the application architecture was mapped onto the remaining operational CLB's and correct operation of the Teramac computer was demonstrated. Teramac did not attempt to detect or correct faults that occurred after the system was initially configured, but the concept can be extended to cover hard faults occurring during system operation.

Comparison of reconfiguration to other hardware FT techniques is made based on the upper limit on p_f , the defect probability per device, that can be tolerated [NSF01, FNS01, LMSP98]. The probability a CLB composed of N_t transistors, each of which fail with probability p_f , functions correctly is

$$P_{clb,w} = (1 - p_f)^{N_t}. \quad (2.24)$$

An arbitrary number, N_c , of CLB's are connected together to form an *atomic fault tolerant block* (AFTB). Since an AFTB can be reconfigured to perform some operation even if one of the component CLB's is faulty, the probability that an AFTB functions properly, $P_{aftb,w}$, is the sum of the probabilities of zero and one CLB failure, or

$$P_{aftb,w} = (P_{clb,w})^{N_c} + (P_{clb,w})^{N_c-1}(1 - P_{clb,w}). \quad (2.25)$$

The failure probability for the AFTB is

$$P_{aftb,f} = 1 - P_{aftb,w}. \quad (2.26)$$

If N_A AFTB's are combined in clusters to perform higher level functions (e.g., adders, memories, etc.), the probability a cluster fails [FNS01] is

$$P_{cluster,f} = 1 - (P_{aftb,w})^{N_A}. \quad (2.27)$$

Suppose R of these clusters are combined into a *supercluster* and the overall computer can diagnose faults in the clusters. If the processor uses the output of a functional cluster, as long as at least one of the R clusters functions correct results are achieved. The probability at least one of the R clusters in the supercluster functions correctly is

$$P_{sc,w} = 1 - (P_{cluster,f})^R. \quad (2.28)$$

The total number of superclusters on the chip, N_{sc} , is

$$N_{sc} = \frac{N_{total}}{R \cdot N_t \cdot N_c \cdot N_A}. \quad (2.29)$$

Finally, the probability the entire chip functions is

$$P_{chip,w} = (P_{sc,w})^{N_{sc}}. \quad (2.30)$$

It is assumed in this example that each cluster can be tested and defective clusters disabled. In reality, limitations on interconnect resources make the problem more complicated. Fault tolerance modelling of reconfigurable systems is an area for further study.

Figure 2.28 shows the performance of reconfiguration versus RMR and NAND multiplexing [FNS01]. In this example, N_c is the number of superclusters in the design, determined by the granularity of the configurable units. The plots show the maximum allowable device failure probability, p_f , versus the level of redundancy, R . The top three lines represent reconfiguration; the middle three lines are NAND multiplexing; and the bottom three lines are RMR. Reconfiguration is able to use devices with failure probabilities several orders of magnitude higher than RMR or NAND multiplexing for any particular level of redundancy. While the model used is fairly simple, it is clear that reconfiguration provides good fault tolerance at levels of redundancy lower than those required by other methods.

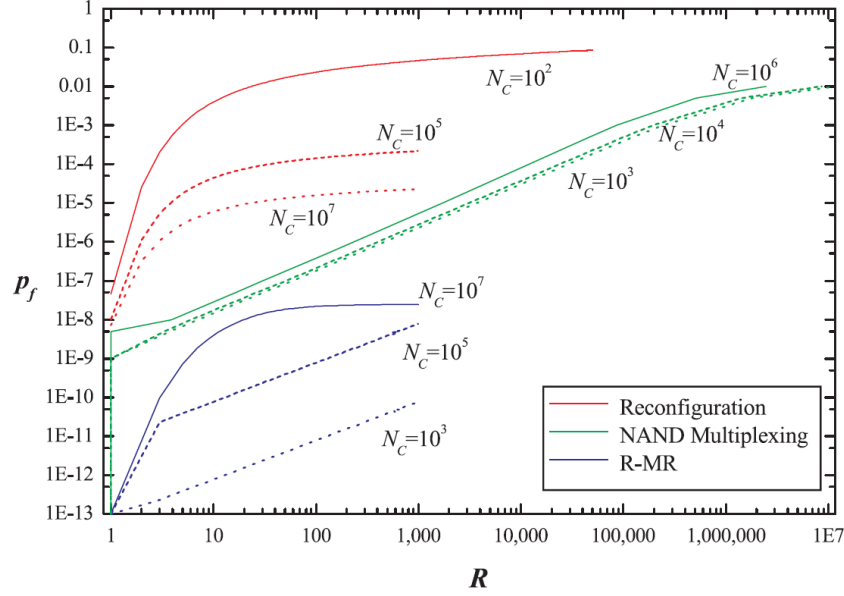


Figure 2.28: Comparison of three hardware FT methods (Max allowable p_f versus Redundancy, R) [FNS01].

Reconfiguration, however, provides no protection against soft errors. For this reason, a fault and defect tolerant computer (FDTC) should implement other fault tolerance techniques in addition to reconfiguration. This research will explore ways to combine these strategies.

2.5 Radiation Effects

Circuit operation can be affected by environmental factors such as cosmic radiation, thermal noise, and power supply fluctuations. These factors sometimes induce errors in digital logic, change the value of bits stored in memory, and cause devices to degrade and suffer permanent failures. In the past, these effects only had a significant impact on microchips used in space applications. But as process size shrinks, these effects are becoming significant even for terrestrial users, and soon computer architects will no longer be able to ignore their impact.

This section examines the common effects of radiation and other environmental factors on the performance of electronic devices.

- Section 2.5.1 examines the causes of radiation-induced faults.
- Section 2.5.2 examines how devices are affected.
- Section 2.5.3 describes how the effects change as the process size decreases.
- Section 2.5.4 examines potential solutions, both at the process and design level.

Due to the large number of storage devices present, Field Programmable Gate Arrays (FPGA) are particularly susceptible to Single Event Upsets and other soft errors. FPGA-specific effects are covered in Appendix A.1.2.5.

2.5.1 Causes. In modern 90nm CMOS, the area per memory bit is a mere $1\mu\text{m}^2$ [KH04]. By 1962, it was predicted that when channel length fell below $1\mu\text{m}$, a single cosmic ray particle strike could short circuit the source and drain terminals, disrupting circuit operation [KH04].

The first errors directly attributed to cosmic rays occurred in 1975, when spacecraft electronics malfunctioned during a “magnetically-quiet time,” meaning the failure was not due to magnetic charging of the spacecraft. By 1978, *soft errors* were observed in dynamic memory devices at ground level. Memory contents had changed some time after being written. Although no damage was observed in the circuit, when new data was stored errors would reappear at different locations. A memory value error due to single particle strike is called a *Single Event Upset* (SEU).

Leading causes of radiation effects include alpha particles and cosmic rays [HSA94]. Cosmic rays are typically protons, but also include alpha particles and heavy atoms with energy levels up to 500MeV. The most common radiation sources in the atmosphere, however, are high energy neutrons. Neutrons are naturally present in the atmosphere and are imparted energy through the impact of cosmic rays and the decay of radioactive nuclei [KH04]. Neutron energies range from 20-300MeV.

2.5.1.1 Alpha Particles. Alpha particles are emitted from a small number of radioactive impurities found in the plastic packages of microchips as well as in the microchip itself [KH04]. Plastics and other materials in the packaging

contain several parts per million of Uranium-238 and Thorium-232. Alpha particle flux rates for current process technologies are about $0.001\alpha/cm^2-hr$ [KH04] and the Soft Error Rate (SER) due to alpha particles does not vary significantly with altitude. SER decreases over time as the impurities decay. SER due to alpha particle strikes is significant—one soft error per day in a 4 kbit DRAM chip, but can be reduced with better packaging materials and shielding between the plastic package and the chip [KH04].

Radioactive impurities are the source of the alpha particles. In 1995, Boron-10 was identified as a significant cause of SEUs [KH04]. It caused as many as 80% of the SEUs in a $0.25\mu m$ SRAM chip. In addition, impurities in the interconnect metals create alpha particles; the silicon wafer itself contains a small number of radioactive impurities.

2.5.1.2 Neutrons. High energy neutrons are generated in the atmosphere by the impact of high energy cosmic ray particles. Neutrons do not carry an electric charge, but when they strike a silicon microchip, the impact energy can cause electron-hole pair formation. The energy required to create a hole pair is dependent on the bandgap of the semiconductor; for silicon it is 3.6eV.

Cosmic rays decrease exponentially with the amount of shielding applied to the chip [KH04]. The atmosphere acts as a natural shield, decreasing SER by three orders of magnitude from aircraft flight altitudes to sea level. Thus, SEUs caused by neutrons are more prevalent in space and aircraft applications than at sea level. Neutron density varies with altitude, being particularly high from 10-40km. The maximum intensity is at 15km. Sea level neutron density is roughly $20\text{ neutrons}/cm^2-hr$ (with energies $> 10MeV$). However, not every neutron striking the surface of the silicon chip strikes a proton. Since the most of the space occupied by an atom is the electron cloud, only one out of 40,000 neutrons striking the surface hits a silicon nucleus in the first $10\mu m$ of depth [KH04].

Table 2.5: Short and long term radiation effects on microelectronic devices.

Transient effects	Source	Method
Rapid annealing of minority carrier lifetime	Particle	Displacement
Transient currents	Particle/Photon	Ionization
Latching conditions in bistable circuits	Particle/Photon	Ionization
Long-Lived Effects		
Increased defect concentration	Particle/Photon	Displacement
Decreased carrier lifetime, mobility, concentration	Particle/Photon	Displacement
Altered population of traps	Particle/Photon	Ionization
Oxidation-reduction reactions	Particle/Photon	Ionization

2.5.2 Effects. Radiation causes both short and long term impacts on electronic devices. This section summarizes several of the most significant effects as observed in materials, transistors, and the system architecture. These effects are common to all ASICs, including FPGAs. The application-level impact on FPGAs is covered in Appendix A.1.2.5

2.5.2.1 Effects on Materials. Several effects relevant to CMOS logic circuits are shown in Table 2.5. The effects are categorized by their duration (i.e., transient or long-term), their source (i.e., high energy particle impact such as alpha particles or neutrons), and the method of damage (i.e., ionization of atoms and resulting hole-pair generation, or physical displacement of atoms in the silicon lattice).

Most long term effects accumulate over time, and are most significant in high radiation environments such as space. Permanent failure of devices can occur due to prolonged radiation exposure. While radiation hardened design is outside the scope of this research, it will be assumed an appropriate level of radiation hardening is used to prevent long term effects. The most significant effects for general purpose applications, at sea level, in aircraft, and in space, are transient effects.

2.5.2.2 *Effects on Devices.* Transient effects in microelectronic devices are caused by concentrated bursts of electric charge generated at random locations in the substrate and collected by the drain diodes in MOSFETs. The charge transfer can be enough to change the logic state of a node in the circuit. A circuit node in 90nm CMOS stores about 1-10fC of charge [KH04]. Alpha particles carrying 3-10MeV correspond to 100fC of charge, well in excess of the amount needed to generate a SEU. Not all the generated charge is collected by the drain. Most of the electron-hole pairs recombine or are collected by reverse-biased PN junctions shorted to the power rail. The fraction of charge collected by the circuit is defined as the *collection efficiency*. The amount of charge necessary to change the output of a device is called the *critical charge*, or Q_{CRIT} .

Depending on their design, circuits are affected by alpha particles and neutrons to different extents. In SRAM cells with lower values of Q_{CRIT} , alpha particles contribute to SER as much as neutrons. For devices with higher Q_{CRIT} values (due to larger device area), neutrons usually dominate. The relative influence of the two types of soft error determines the overall SER for the device.

The electrical pulse generated by the impact of an alpha particle or neutron may not affect the output state of the local transistor, a higher level logic gate, or the system. A typical single event upset (SEU) lasts about 100ps, and if the charge disturbance is less than the noise margin for the device the charge pulse will not effect the output of the logical gate. If it exceeds the noise margin, it can cause an inverter to change state. If connected in a feedback loop such as in a latch or flip flop, the error in the first inverter is passed on to the second inverter, changing the memory state of the device.

In a combinational circuit, many transient effects dissipate prior to the end of the clock cycle and the latching of results and are said to be masked. Therefore, the soft error rate (SER) observed at the module or system level is often less than the

SER at the device level. The higher level SER is *derated* to remove soft errors that are masked. Three common types of masking are

- Logical masking occurs when the output of a gate is controlled by the input not subject to the soft error. For example, a NAND gate with inputs of ‘0’ and ‘1’ would not be affected by a soft error on the ‘1’ input since it does not change the output.
- Temporal masking occurs when the noise on the input of a latch or flip-flop is outside of the clocking window and does not change the state of the memory.
- Electrical masking results from the limited bandwidth (i.e., switching speed) of devices. Transients with bandwidths greater than the cutoff frequency of the device are attenuated and the pulse amplitude visible on the output of the transistor may be reduced below the threshold of the next device in series. Thus the effect may be limited to a few serial logic gates.

2.5.2.3 Effects on Systems. Many soft errors are masked and do not affect the output or state of the overall system. The soft errors may or may not be detectable, and may or may not be correctable. For example, many modern memories contain parity bits to detect errors. Some memories implement error correcting codes to correct single bit errors. The following SERs have been observed in memory devices [KH04, Xil03]:

- Neutron induced soft errors in a 256kbit SRAM on a commercial aircraft resulted in an SER of 1 error per 80 days.
- Alpha particle strikes at a rate of several per $cm^2 - hr$ led to an SER of 1/day in a 4kbit DRAM.
- A supercomputer with 156Gbit of DRAM failed several times per day.
- The SER in pacemakers is about the same errors caused by background neutron radiation.

- The European Space Agency’s Freja satellite experiences > 200 SEUs/day. More than 40 proton-induced latchup events were observed in three years in orbit.

Logic errors are more difficult to detect and correct and will soon dominate chip level SER in ASICs [KH04]. Testing of real processors has not yet shown this to be a major problem, but with further process scaling, soft errors in logic will soon become more prevalent.

2.5.3 Relation to Process Scaling. The impact of process scaling on soft error rate is difficult to predict as it depends on many competing factors. Many studies have been done to determine the effect of voltage and size scaling on SER and to find ways to keep the overall SER down. Even if the bit level SER is kept down, the exponential growth in the number of devices on a chip can result in a system level SER that grows as process size shrinks.

At sea level, scaling effects in SER are an aggregate of alpha particle, high energy neutron, and thermal neutron effects. Power supply voltage is also a factor. SER has been shown to increase by a factor of two when power supply voltage was decreased from 1.2V to 0.8V [KH04]. Device size plays a role, as the critical charge, Q_{CRIT} , decreases by K^2 with the constant scaling rules. At the same time, the collection area decreases with size, lowering SER. The collection efficiency decreases with increased substrate doping and reduced bias voltages, decreasing SER.

Theoretical predictions show SER due to alpha particle strikes remains relatively constant with scaling. Experimental measurements have produced minor variations, but in general agree with the theoretical predication.

For neutrons, experimental results vary widely. For example, one study shows a 50% decrease in SER in SRAM cells from $0.5\mu m$ to $0.25\mu m$, but a 300% increase from $0.25\mu m$ to $0.14\mu m$ [KH04]. Another study reports an increase of 8% per process generation from $0.25\mu m$ to 90nm [KH04]. This variation may be explained by disproportionate scaling of Q_{CRIT} with respect to collector efficiency.

Technology trends, then, indicate a moderate increase in SER/bit or SER/latch with process scaling [KH04]. Due to careful modeling of SEUs, process improvements, and device hardening, this trend has not been manifest in production devices. Whether SER can be held constant with scaling remains to be seen.

2.5.4 Solutions. Several strategies can overcome the effects of soft errors on digital circuits. The fabrication process can be controlled to reduce the numbers of radioactive impurities. Better modelling and characterization techniques can be developed to accurately model the impacts of design and process changes on SER. Finally, architectural changes can be incorporated to detect and overcome faults.

Many process improvements have already been incorporated to reduce the soft error rate. When Boron-10 was discovered to be a cause of soft errors, boro-phospho-silicate glass was removed from the fabrication process. In addition, increasing the purity of silicon wafers will reduce the number of radioactive impurities.

The use of Silicon On Insulator (SOI) also reduces soft error rate. SOI devices have lower junction capacitances and better noise isolation since the thinner substrates present a lower collection volume and thus collect less charge during a particle strike. It has been shown that SER can be reduced by a factor of five or more through use of SOI. Further radiation hardening techniques can decrease SER by as much as 100 times. Care must be taken in design, however, as the forward biasing of the substrate is more significant in SOI, which creates a parasitic bipolar transistor whose signal can be amplified in the circuit and contribute to soft errors. In the worst case this can cause SER to exceed bulk CMOS.

It is also important to be able to accurately quantify and model the effects of radiation on devices. Accurate design trade-offs cannot be made without an understanding of the impact of design changes on SER. Most radiation modelling has been targeted toward bulk CMOS using conventional materials. As new materials such as copper interconnect, high k dielectric materials, and silicon-germanium strained silicon come into use, modelling of SEU effects becomes more difficult. Some of these

materials increase SER. First order modelling shows neutron SER increases with the mass density of the materials present. Copper, tantalum, tungsten, and cobalt in CMOS fabrication may increase SER by a factor of two or more [KH04].

Simulation of SEU effects is difficult, with complexity growing exponentially with the size of the circuit. Even testing is difficult to perform as the impact rates of alpha particles and neutrons is very low. Accelerated testing is often performed by bombarding a microchip for a short time with white neutron or proton beams at a higher rate than found in the environment. Results are then scaled with time to match a lower radiation rate for a long time period.

Architectural changes can make the circuit more tolerant to soft errors. Many memory designs incorporate parity bits and error correcting codes (ECC) to detect or correct single bit errors in memory arrays. ECC is very effective, as relatively few SEUs result in the upset of two or more adjacent memory bits. Interleaved memory designs can separate bits checked together by ECC, reducing the probability that of a multi-bit error.

2.6 Fault Tolerant Architectures

2.6.1 Fault and Defect Tolerant Systems. This section examines several recent experimental and hypothetical system architectures incorporating fault tolerance. These systems can be broadly classified in four categories:

- Error Tolerant Systems
- Defect Tolerant Systems
- Fault Tolerant FPGA-Based Systems
- Array-Based Multiprocessor Systems

2.6.1.1 Error Tolerant Systems. Certain applications do not require accurate computation [BGM04]. For example, video compression and some signal processing algorithms produce approximations rather than exact results. In many

cases, errors in a computation result is acceptable, since it would go unnoticed amid the inaccuracies inherent in the application algorithm.

This type of application may be well suited for error-prone chips implemented with molecular crossbars or other devices. Errors in the control logic in the circuit would still be unacceptable, as the processor may lock up or crash. Likewise, errors in data-producing module such as adders and multipliers would be unacceptable when producing results for the control flow of the program. But errors in the data output may be tolerated. In these cases, the system designer can trade off performance versus reliability, resulting in a design that operates very fast (either singly, or using many processors in parallel on a dense chip) at the cost of accuracy.

2.6.1.2 Defect Tolerant Systems. Teramac, constructed by Hewlett-Packard in 1998, is a defect tolerant computer [HKS98, Cla98]. Teramac is a massively parallel computer constructed from inexpensive, defect-prone FPGAs. Constructed using 864 identical FPGAs, only 217 of the chips passed fabrication testing. The remaining 75% were provided for free by the manufacturer, as they would otherwise have been discarded as defective. In all, over 220,000 defects were identified in the FPGAs. After constructing the computer from the 864 FPGAs, the system was powered up and a test configuration was loaded onto the FPGAs. Defect locations and types (logic and interconnect) were mapped and provided to a specially-built application compiler, which placed and routed the final multiprocessor architecture onto the FPGAs. Teramac demonstrated correct operation using defective components.

The Teramac project showed it is possible to build a powerful computer from defective components, given sufficient routing resources. It was also shown the communications resources do not have to be regular in structure, so long as they provide a sufficiently high degree of connectivity. Finally, Teramac identified interconnect resources as the most critical aspect of the design [HKS98]. Future molecular computers will require tremendous amounts of communications bandwidth to connect the various parts of the circuit.

2.6.1.3 *Fault Tolerant FPGA-Based Systems.* Space-Based Radar is a

fault tolerant systems being developed using FPGAs. Designed by the Jet Propulsion Laboratory and the Air Force Research Laboratory, the system will provide onboard processing of radar data [LCC⁺04], as well as:

- Provide a capability to reconfigure the FPGAs to support algorithm updates after launch.
- Provide graceful degradation of capabilities while operating in a radiation-intensive environment for a period of at least three years.
- Be tolerant of the space radiation environment, while achieving a given reliability and availability.

The architecture of the processor is a tightly-coupled reconfigurable computer consisting of a FPGA front end connected to a digital signal processor back end. For fault tolerance, the system incorporates modular TMR, replicating the entire FPGA processor three times and comparing the results. A separate Fault Management Unit controls the operation of the three FPGA processors, as well as implementing periodic scrubbing of the configurations, and periodic off-line testing of the FPGAs. In addition, a failed FPGA can be disabled completely.

Future design iterations of the system will include redundancy at lower levels (i.e., circuit, gate, and module levels). Other fault tolerance techniques under consideration include algorithm-based fault tolerance, time redundancy, and information redundancy [LCC⁺04].

The Center for Reliable Computing at Stanford University is developing a two FPGA architecture that provides error detection and autonomous self-repair without external intervention [MHS⁺04]. Each FPGA design includes internal Concurrent Error Detection (CED) circuitry. Many faults and errors can be corrected internally using standard fault tolerance techniques. For suspected configuration bit errors due to SEUs, one FPGA reads and compares the configuration of the other. Scrubbing techniques are used to repair SEUs in the configuration.

Permanent fault recovery is provided by relocating portions of the application design to unused columns in the FPGA. Pre-compiled alternate configurations are loaded at random, moving columns to different locations. The operation producing the error is retried using different configurations until the error disappears. The column not used in the final configuration is deemed the defective column. This method of fault diagnosis claims to be simpler and faster than other techniques such as Roving Self-Test and Repair (STAR) [MHS⁺04] (cf., Section 2.6.3.3).

A column relocation strategy limits the number of defects that can be tolerated. When a permanent fault is detected, an entire column is marked defective, which limits the number of faults that can be repaired. If the FPGA contains M column, with N columns being used by the application, only $M - N$ spare columns exist. As few as $M - N$ faults (i.e., one per column) can render the system unusable. This is a limitation primarily due to the column-based reconfiguration architecture of the Xilinx FPGAs used by the project. Alternate FPGA designs can remove this limitation.

Another limitation of the approach is the pre-defined alternative configurations, which target faults in the CLBs rather than in the interconnect. The alternative configurations must pass signals across unused columns. If the fault lies in one of the configuration memory cells in a switchbox matrix, in a pass transistor, or is in the wiring itself, the faulty component may be used by the alternative configuration, even though the column itself is not used. This is an area that needs to be addressed.

2.6.1.4 Theoretical Array-Based Systems. Some systems are based upon massively parallel multiprocessor systems with configurable interconnect networks. A multi-stage interconnect network was proposed by [ACD⁺02]. In this system, the processing nodes are FPGAs, connected in a large multi-stage switch network containing a high level of redundant links and two levels of fault tolerance. System level fault tolerance removes a failing FPGA node from use and transfers its function to another node. The switch network is reconfigured to route the inputs and outputs from the old node to the new one. The second layer of fault tolerance reconfigures

the FPGAs themselves. Each FPGA provides spare CLBs in each column. Alternate FPGA configurations are used to move the application design from faulty to working CLBs. As a final option, entire columns can be moved [MHS⁺04].

2.6.2 Fault Tolerant FPGAs. This section examines research efforts to construct more reliable FPGAs. Fault tolerance techniques have been proposed as methods to overcome manufacturing defects, thereby increasing yield, as well as making operational FPGAs less subject to SEU-induced errors and other failures.

One of the first papers to address the problem of increasing yield in FPGAs through redundant resources was [HTA94]. Although the research focused on CMOS FPGAs, several key challenges were identified that will prove equally relevant to future fault and defect tolerant computers implemented with other device technologies. The most important is the burden placed on the users of the chip. The capabilities and performance of each chip will vary greatly depending on the number, type, and locations of defects. Each device must be individually tested after fabrication to determine functionality, and whether the intended application design can be mapped to the defect-free resources. Yield will be much more difficult to predict than in current ASICs, as single defects will no longer render the entire chip useless.

The standard yield equation for ASICs [HTA94] is

$$Y = \left(1 + \frac{\bar{\lambda}}{\alpha}\right)^{-\alpha}, \quad (2.31)$$

where α is a *clustering parameter* to account for the fact that defects often occur in close proximity on the wafer. This factor produces higher yields than would be expected from a uniform distribution of defects. The mean number of defects in the chip area, $\bar{\lambda}$, is

$$\bar{\lambda} = A \times d \quad (2.32)$$

where A is the chip area, and d is the defect density (in defects per unit area).

FPGAs with one or more defective resources (i.e., either CLBs, switching elements, or configuration memory cells) can still be used. Thus, it is more appropriate to measure *cell yield*, the fraction of FPGA CLBs that are usable [HTA94]. In the simple cell yield model, the primary resource under consideration is the CLB, while interconnect resources are assigned to the CLB cells. Depending on where a defect occurs, it can disable one or more cells (i.e., CLBs and associated interconnect) in the FPGA, since FPGAs contain interconnect resources that span multiple cells. An interconnect fault affecting one or more column lines may disable an entire column of CLBs. The probability a defect disables an entire column or row of cells is P_{col} and P_{row} . Likewise, a defect affecting the configuration or power distribution portions of the FPGA can disable the entire device. The probability a defect disables the entire array is P_{array} , while the probability that an occurring defect disables a single cell is P_{cell} . These probabilities are relative to each other, with $P_{cell} + P_{row} + P_{col} + P_{array} = 1$. Thus, cell yield is

$$\bar{\lambda} = [A_{cell} \cdot (i \cdot j \cdot P_{array} + i \cdot P_{row} + j \cdot P_{col} + P_{cell}) + A_{overhead}(i, j)] \times d, \quad (2.33)$$

where A_{cell} is the area of a single cell (i.e., the CLB and its portion of the interconnect matrix), $A_{overhead}(i, j)$ is the area of configuration logic for the entire FPGA, and d is the defect density (in defects per unit area) [HTA94].

Cell yield can be increased using segmented or hierarchical interconnects. By breaking row and column lines into segments, a fault on one segment affects fewer CLBs. This has the effect of lowering P_{row} and P_{col} . For this reason, segmented routing will continue to be included in future FPGA designs as fault tolerance becomes more of a consideration.

Another design consideration relevant to fault tolerant FPGAs is the channel design for routability [RN95]. A method for finding the optimal channel design (i.e., channel widths and segmentation) was developed using simulated annealing. By comparing the fault tolerance of an FPGA architecture synthesized for routability and

performance to that of an FPGA architecture designed for improved fault tolerance using the new technique, a 6% increase in the number of faults that could be tolerated was realized.

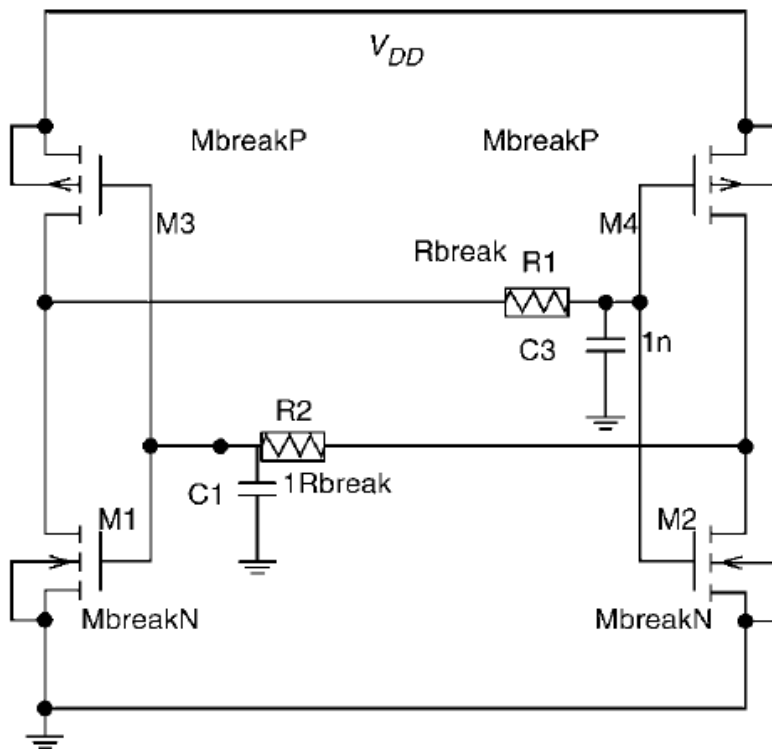
The remainder of this section addresses the following categories of improving reliability in FPGAs:

- Circuit level improvements,
- Logic block designs,
- Fault tolerant interconnect,
- Spare logic blocks interspersed in the array, and
- Spare rows and columns of logic blocks.

2.6.2.1 Circuit Level Improvements. Xilinx produces a line of radiation hardened FPGAs that include a thin epitaxial layer in the fabrication process to reduce the susceptibility to Single Event Latchups [Xil03]. A variety of VLSI layout techniques are used in radiation-hardened circuit design to provide increased protection at the cost of special materials, increased design times, and larger circuit size. Radiation hardened processes typically lag behind conventional processes by one or two generations, so radiation-hardened FPGAs provide less capability than commercial FPGAs.

Rather than applying these techniques to the entire device, several papers have proposed alterations to the VLSI layout of the FPGAs configuration cells as a method of increasing the resistance to SEUs [Wan04, SGV⁺04]. This method is sometimes called radiation “hardening by design.” This method allows the use of conventional CMOS fabrication processes.

Radiation-hardened memory cell designs use resistor-capacitor pairs to filter Single Electron Events (SEE) [Wan04]. As shown in Figure 2.29, resistors R1 and R2 are added in series with the gate capacitances of the two inverters (M1-M3 and M2-M4). The resistor-capacitor pairs form a filter on the input lines entering the inverters,



filtering out the high frequency components of a SEE. This technique does not require redundant transistors, but may require large resistors. As process size shrinks and node capacitance decreases, resistor values must increase to maintain the correct RC time constant to filter the high frequency components of a SEE. For example, in $0.25\mu m$ processes resistance values in the megaohm range are necessary, and the addition of these large resistors may cause an unacceptable increase in switching delays due to the larger RC time constant and the increased time needed to charge the gate electrodes of each transistor [Wan04].

A second technique observes that the typical configuration bitstream of an FPGA is composed of 87% zeros [SGV⁺04]. Thus, the design of the SRAM memory cell can be optimized to be more resistant against $0 \rightarrow 1$ SEUs than $1 \rightarrow 0$. The new SRAM cell is called *Asymmetric SRAM* (ASRAM). The ASRAM-0 cell design has a lower leakage current and increased soft error immunity when storing a ‘0’ bit. In an

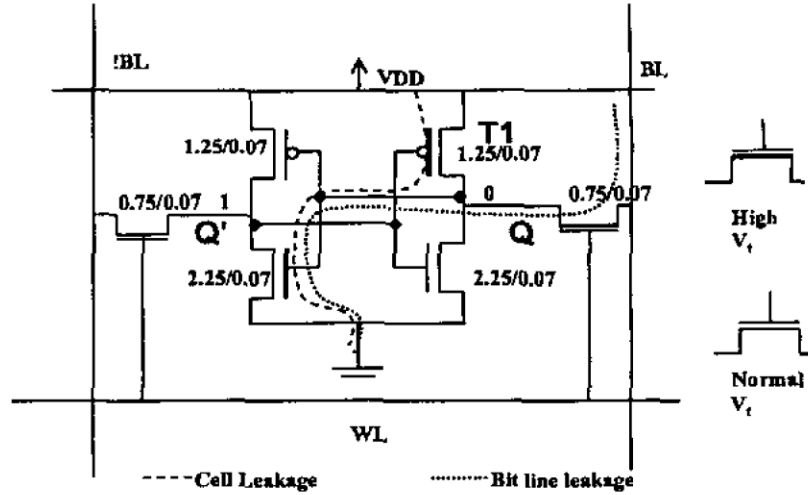


Figure 2.30: The Asymmetric-0 SRAM cell provides increased SEU protection when storing a 0 bit [SGV⁺04].

ASRAM cell, the threshold voltages, V_t , of the transistors are chosen to minimize the leakage current in the ‘usual’ state. An ASRAM cell is illustrated in Figure 2.30. The circuit structure is the same as a standard memory cell. The upper right and lower left transistors have a higher threshold voltage than the other two transistors in the feedback loop. When storing a ‘0’, the node labelled Q is at $V_{ss} = 0V$. Thus, there may be a small leakage current from V_{dd} across the upper right transistor to node Q. Likewise, node Q' is at V_{dd} . A small leakage current exists from node Q' to V_{ss} across the lower left transistor. Increasing the threshold voltage for these two transistors will reduce the leakage current at the expense of a small performance penalty. The ASRAM-0 cells used in the experiment reduced leakage energy consumed by the configuration SRAMs by a factor of 18 compared to standard balanced SRAMs.

A final design technique incorporates redundant transistors to provide immunity to single event upsets [CNV96]. The design is called *Dual Interlocked Memory Cell*. The concept is illustrated in Figure 2.31. In this design, redundancy in the memory latches stores a second copy of the data state. Thus, one latch provides a “state restoring feedback” function to the other in case of an SEU. In this diagram, IA1 and

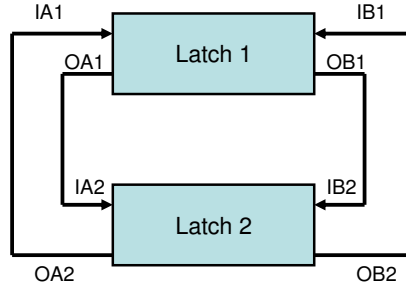


Figure 2.31: The conceptual view of the DICE Memory cell [CNV96].

IB1 correspond to the D and \overline{D} inputs to latch 1, while $OA1$ and $OB1$ are the Q and \overline{Q} outputs.

The circuit diagram for the DICE SRAM cell is shown in Figure 2.32. The four nodes $X_0 - X_3$ store the state of the cell. Logic ‘1’ is ‘1010’, and logic ‘0’ is stored as ‘0101’. Transistors N4-N7 form transmission gates to enable or disable read/write access to the cell. The design of the feedback loops is such that a SEU occurring at one of the nodes X_i can temporarily affect the logic state of X_{i+1} (in the case of a negative upset pulse, converting a $1 \rightarrow 0$) or X_{i-1} (in the case of a positive upset, converting $0 \rightarrow 1$). However, the SEU will not affect the logic state stored in the other feedback loop. Thus, the other two nodes are isolated from the effects of the SEU. The logic perturbation is removed after the transient ends due to the state-reinforcing feedback function of the other two nodes.

The DICE cell protects against a SEU affecting only one node but requires 12 transistors versus the six of a standard SRAM cell. If the particle impact generates pulses at two nodes in the same logic state (e.g., X_1 and X_3), the immunity is lost and an SEU occurs. This probability can be kept low if the transistor drain areas of the two nodes are suitably spaced on the VLSI layout. This will become difficult as process size decreases, however.

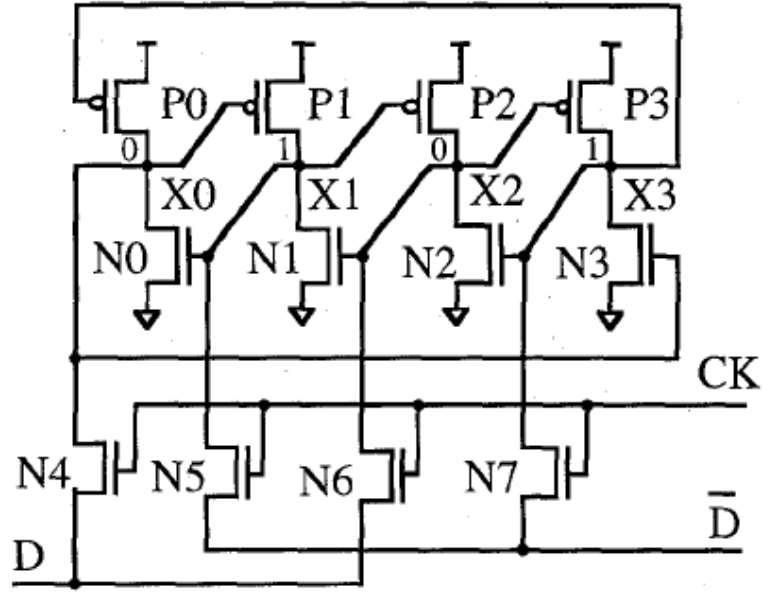


Figure 2.32: DICE Memory cell is immune to SEUs [CNV96].

2.6.2.2 Logic Block Designs. Over the years, many researchers have investigated changing the FPGA configurable logic block to provide the maximum flexibility and functionality with the minimum amount of overhead (i.e., redundant interconnection). Over time, the four-input lookup table logic block has become standard commercial practice. More recently, alterations to the CLB structure to increase fault tolerance have been proposed. This section highlights several of these projects.

The Field Programmable Transistor Array (FPTA) can be viewed as a FPGA with extremely fine granularity; the hardware is configurable at the transistor level. The FPTA array of transistors is interconnected by programmable switches [Sto99, KZJS00, SZK⁺01]. More interconnect resources are provided than in a larger grained FPGA, although not every possible connection is possible. A proposed node architecture for the FPTA is shown in Figure 2.33. The FPTA cell consists of eight transistors and 24 programmable interconnect points. The cell corresponds to the

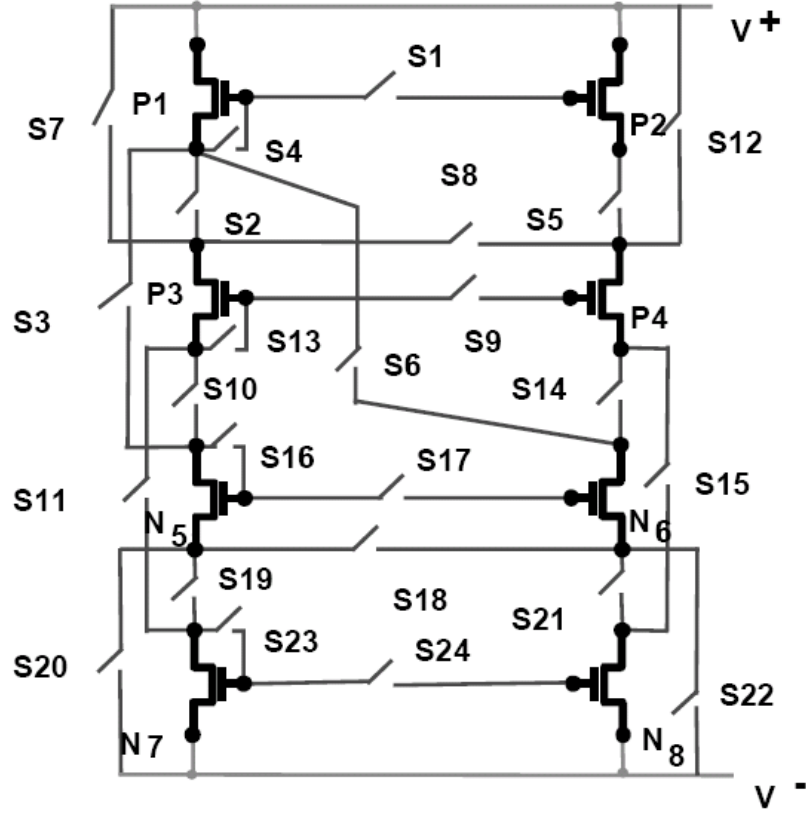


Figure 2.33: Node design for a Field Programmable Transistor Array. Similar in function to an FPGA, the FPTA has a much finer granularity [Sto99].

CLB in a FPGA. The FPTA consists of an array of cells in surrounded by a mesh of programmable interconnect.

Due to the fine granularity, the FPTA is more flexible than a FPGA and capable of implementing a wider range of application circuits. On the other hand, the FPTA has a much higher interconnect and configuration overhead. If the 24 programmable switches in Figure 2.33 are implemented using four transistors each, 96 transistors would be required to control the eight application transistors. While an example of a FPTA has been fabricated for research purposes, the tremendous overhead required makes it unlikely the FPTA will be widely adopted for CMOS processes. The FPTA architecture may prove useful with technologies such as molecular switches, which can implement the memory bits very efficiently.

Table 2.6: Stuck-Open and Stuck-Closed transistors in the multiplexer can be detected using the voltage level of the output. A simple voltage detector is formed from two inverters with different threshold voltages [PCL⁺02]. A fault is indicated when Inverter 1 = V_{DD} and Inverter 2 = GND.

V_{out}	Inverter 1	Inverter 2
$GND < V_{out} < V_{T1}$	V_{DD}	V_{DD}
$V_{T1} < V_{out} < V_{T2}$	V_{DD}	GND
$V_{T2} < V_{out} < V_{DD}$	GND	GND

Another fine-grained CLB design has CLB granularity at the level of individual gates [SP03]. The architecture is designed to support research in evolutionary algorithms (also known as genetic algorithms). Evolutionary algorithms are used with partial reconfiguration to progressively modify the application circuit to improve performance (and recover from faults). While research is ongoing, current results indicate genetic algorithms might be useful for fault tolerance. In addition, fine-grained FPGAs provide more flexibility and better fault recovery capability than coarser-grained (i.e., LUT-based) CLB designs [SP03].

A third modification to the CLB design detects transistor faults in the CLB [PCL⁺02]. As shown in Table 2.6, a simple circuit detects abnormal voltage levels that signify stuck-closed and stuck open faults in the transistors of the multiplexers in the CLB. The checking circuit is constructed from two inverters connected in parallel to the output of the multiplexer. The two inverters use transistors with specially selected threshold voltages. LUT memory faults are detected using a built-in current sensor to detect anomalous current flows.

This technique can detect single transistor stuck-open or stuck-closed errors. A simple CLB design used in experiments consisted of 300 transistors. Ninety-six additional transistors were required to implement the fault detection circuits, for an overhead of 32%.

2.6.2.3 Fault Tolerant Interconnect. The use of extra interconnect resources to provide fault recovery has often been proposed [HTA94, HD98, HTL04b, HTL04c, HTL04a, HSN⁺93]. Sufficient redundant interconnect can eliminate the need to relocate CLBs when a fault occurs in interconnect lines or switch matrices [HD98]. It may even be possible to switch from faulty interconnect lines to non-faulty resources without using a router.

2.6.2.4 Spare Logic Blocks. Fault tolerance can also be achieved using spare configurable logic blocks scattered throughout the FPGA. The redundant CLBs are no different from the other CLBs, but are not normally used by the placement software during initial placement of the application design. When a failure is detected in a CLB, the spare is activated. Depending on the location of the spare relative to the faulty CLB, the configuration contained in the faulty CLB is either transferred to the spare, or multiple CLBs are shifted to new locations. Depending on the design of the FPGA and its interconnect structure, the routing of signals between the CLBs may change substantially. Simple moves are accomplished by shifting the appropriate configuration bits, while more complex routing changes will require a router.

Topologies for the placement of spares include placing a spare CLB at the end of each row and column of the array [KI94, HD98]. This idea was extended to include a torus structure by connecting the end of each row back to its beginning, and the top of each column to its bottom [DI01]. A 2x2 node covering structure requires no modification of the configuration bit files, while switching of the the wiring internal to the nodes is done automatically to replace a faulty CLB inside the node [Els03].

Two other approaches include “king-shifting” and “horse-shifting” algorithms, named after chess pieces [DKI99]. As shown in Figure 2.34, king-shifting places a spare at the center of a 3x3 cluster. It can be used to replace any of the eight surrounding CLBs. In horse-shifting, the spare can only replace vertically or horizontally adjacent cells. In terms of the number of spares required, king shifting is more efficient, since

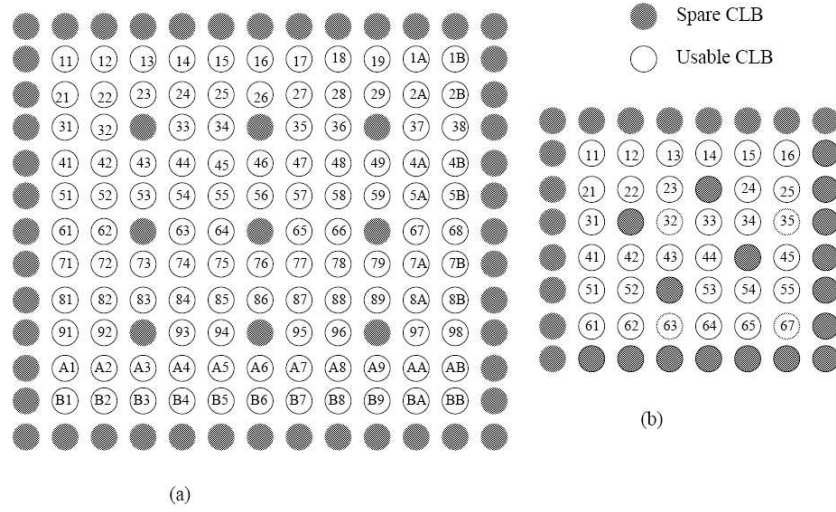


Figure 2.34: Spare CLB locations for king shifting (a) and horse shifting (b). In king shifting, the spare can replace any adjoining cell, while in horse shifting, the spare can only replace vertically or horizontally adjacent cells (no diagonals) [DKI99].

one spare is used for every eight usable cells, while in the horse method, one spare is used for every five cells.

2.6.2.5 Spare Rows and Columns. Redundant CLB arrangements also include sparing entire rows or columns [HSN⁺93, HTA94, MHS⁺04]. The column relocation method is attractive because it can be implemented with current commercial FPGAs, which allow partial reconfiguration by columns (See Section A.1.3). In a typical scheme, each alternate configuration is pre-routed, and the partial bit files are stored [MHS⁺04]. When a CLB failure is detected, the application module using the column containing the faulty CLB is moved to the spare column. The system uses pre-routed alternate configuration. This scheme is described in more detail in Section 2.6.1.

This section describes numerous techniques for post-detection fault recovery. For these techniques to be useful, it must be possible to both detect the error, diagnose the cause, and identify the failed resource (i.e., CLB, interconnect, switch matrix,

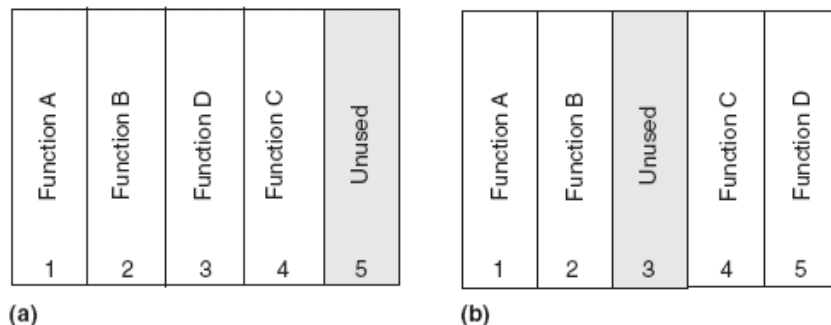


Figure 2.35: With column shifting, the original configuration places a spare column to the right side of the array (left). Following a fault in column 3, function D is moved to the spare column (right). A variation on the technique would shift both D and C to the right [MHS⁺04].

CMC, etc). Methods for detecting and diagnosing faults in FPGAs are described in the next section.

2.6.3 FPGA Testing. As with other forms of VLSI testing (cf., Section 2.4.1), FPGA testing has become more complicated and time consuming as the number of devices on the chip increases. Typically, chips undergo manufacturing tests at the completion of the fabrication process. A sequence of test vectors is applied to each chip to detect functional and parametric faults. With conventional *fault detection*, chips containing faults are discarded.

The architecture of a FPGA is by design redundant, containing a large array of regular structures. Even with many faults, the remaining resources on the FPGA can be used, albeit at a reduced capacity. Therefore, defect tolerance is a way of increasing yield, and *fault diagnosis* becomes as important as *fault detection*. Fault diagnosis locates and characterizes the detected fault, allowing the FPGA design software to avoid the defective resources (e.g., CLB, switch element, interconnect) during application placement and routing.

FPGA fault detection and diagnosis can also be done during operation. FPGAs used in space and other radiation intensive applications degrade and fail during

operation. Fault diagnosis enables the continued use of the remaining resources in the FPGA. This approach has already seen use in long term space missions [Rat04]. When a fault has been diagnosed, a new bit file for the application design is generated avoiding the defective resource and sent to the spacecraft via radio.

Many of the test strategies discussed in this section take advantage of the FPGA's reconfigurability to support the test process. Unlike ASICs, which have a fixed, limited amount of logic available to support testing, the FPGA can implement much larger test structures in the CLB array, which are subsequently overwritten by the application circuit. Fault and defect tolerant computers of the future will likely include reconfigurability, and have similar capabilities. Thus, the test techniques described in this section will have application to not just modern FPGAs, but future fault and defect tolerant computers based on non-CMOS technologies.

The remainder of this section is divided into three parts:

- Classification of test approaches and common fault models.
- FPGA fault detection.
- FPGA fault diagnosis.

2.6.3.1 Classification of Approaches. Test engineers want to achieve maximum test coverage with minimal testing time. Exhaustive testing, whereby every possible input combination is tested against every state of the state machine, is impossible with large devices. A good understanding of how the devices are likely to fail allows the test engineer to design an efficient test set that provides the best coverage with the fewest number of test vectors. A new dimension is added for FPGAs: the need to test with the fewest configurations of the FPGA.

An excellent overview of FPGA test approaches is contained in [DI03]. FPGA test engineers use a variety of fault models. Academic researchers, in contrast, are somewhat limited in their ability to develop accurate circuit level fault models, as FPGA vendors do not typically provide details of the internal structures of CLBs and

the circuits used to program and control FPGA operation [DI03]. Therefore, many academic studies in this field use hybrid models combining functional fault models and stuck-at fault models. When detailed knowledge of a FPGA resource is not available, functional fault models represent a resource as a black box and detect errors at the digital level. Where more specific knowledge is available (e.g., interconnect resources are typically described in detail by FPGA vendors), more detailed fault models can be used. In this case, stuck-at, open circuit, short circuit, and bridging faults are often used.

FPGA testing is typically divided into two stages: fault detection and fault diagnosis. Fault detection detects errors caused by faulty FPGA resources. Fault diagnosis localizes the fault to a specific resource that can be marked and avoided. Testing is done at the time of manufacture as well as during use in an application system. In-system testing can be done *off-line*, during which operation of the application circuit is suspended, or *on-line*, which is done while the application circuit continues operation.

A good fault detection/diagnosis approach should have the following qualities:

- Maximal test coverage.
- Fewest number of test vectors and test configurations of the FPGA.
- If possible, method should target current FPGA architectures, without requiring HW changes.
- If HW changes are used, they should minimize HW overhead such as extra configuration memory.
- Should not assume that certain resources are fault free (i.e., do not test CLBs, assuming that the interconnect is fault free).
- Faults in each different resource type should be included (i.e., in addition to CLBs, also consider interconnect, switch matrices, configuration memory, I/O blocks, programming circuits, etc.)

2.6.3.2 Fault Detection. Fault detection in FPGAs can be divided into three categories [DI03]:

- Testing by reconfiguring the FPGA to implement test circuits.
- Testing by modifying the FPGA’s architecture.
- Parametric testing using variations in timing and current consumption to detect faults.

The most common test approach is the first method, which uses test circuitry created by configuring the FPGA’s programmable logic. Several papers propose modifications to the FPGA architecture to better support this type of testing. Few researchers have examined I_{DDQ} testing, which uses variations in the power supply current, I_{DD} , to detect short circuits, slow switching speeds, and other problems that may not cause logic errors detectable by normal means.

Fault detection typically targets each of the major resource types on the FPGA. The tester must verify the correct operation of each of the CLBs, interconnect lines, switch matrices, I/O blocks, configuration memory cells, and configuration control circuitry.

Most approaches to CLB testing examine a single CLB and then repeated for the entire CLB array. Exhaustive testing of a CLB requires an unacceptable number of test vectors (i.e., 2^{I+O+C} , where I is the number of input lines to the CLB, O is the number of outputs, and C is the number of configuration bits in the CLB). Therefore, minimizing the number of test configuration used is critical [HL96, SKCA96]. Since FPGA manufacturers do not publish the details of their CLB designs, most academic studies of CLB testing use functional fault models. Different numbers of “minimum test configurations” have been proposed, from 21 [HL96] to four [WT99]. The effectiveness of these test approaches depends on how accurately the assumed fault model models real faults in the CLB.

When testing the entire array of CLBs, most techniques assume the interconnect resources have been tested and are reliable [DI03]. The most basic test approach tests each CLB in sequence, connecting its inputs and outputs directly to the pins of the FPGA [HL96]. Test vectors are supplied externally. Multiple CLBs can be tested in parallel, limited by the number of I/O pins available.

Rather than controlling tests from an external source, *Built-In Self Test* can be used [SMSP97, SKCA96]. Unlike conventional BIST, which incorporates custom hardware onto an ASIC to support testing, FPGA BIST implements the test controller with programmable logic. A portion of the CLB array is configured as the test controller, and tests other CLBs in the array. The advantage of this BIST approach is testing can be controlled on-chip, with no additional test-specific hardware resources. However, a highly flexible interconnect structure is required to adequately test all of the configurable resources.

Testing of interconnection resources has not received as much attention as logic blocks. Interconnect testing falls into two categories: BIST-based testing, and non-BIST testing. Interconnect BIST is similar to logic BIST [SWHA98]. Some CLBs are configured as test pattern generators, while others analyze the outputs of the devices under test (DUT). Several configurations of the test must be done in sequence to test the entire FPGA interconnect fabric. Of course, BIST-based testing requires component CLBs to be fault free.

Non-BIST based testing of the interconnect is controlled from an external source [RPFZ98]. Renovell proved only three configurations are needed to test for single faults in a switch box, as shown in Figure 2.36. In this simple switch box, two wires enter the switch from each direction. The only allowable connections between lines labelled '1' are with other '1' lines. The '0' lines can only be connected to the other '0' lines. Test configurations for larger switch boxes are generated in a similar manner. Real FPGAs have more complicated switch box designs. In addition to the

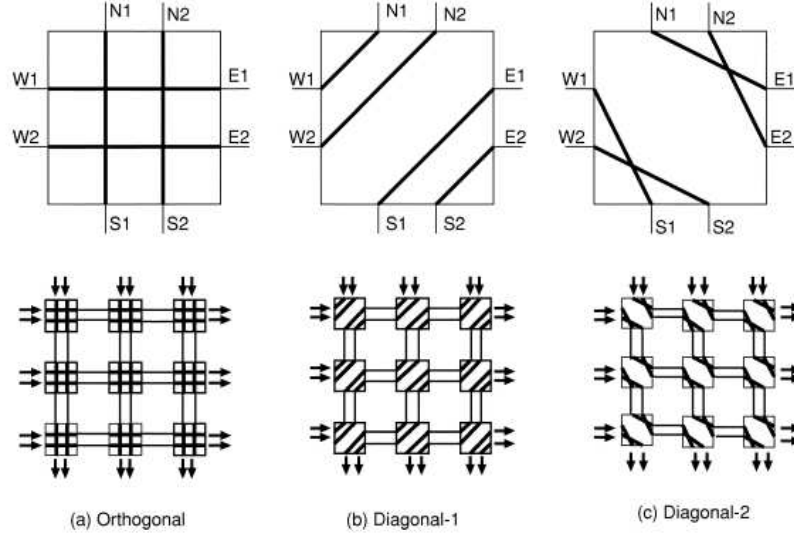


Figure 2.36: Renovell showed that only three configurations are needed to test a switch block for single faults [RPFZ98].

interconnect lines and switch boxes, the connecting blocks linking the interconnect with the CLBs must be tested as well [RPFZ99].

Modifications to the FPGA architectures better support fault detection. *Configuration shifting*, for example, moves test structures from one area of the FPGA to another, testing portions of the FPGA in sequence. Likewise, the configuration memory cells in the CLBs linked in series allow the configurations of one CLB to be shifted to the next [DI99, DI00, DI01]. Thus, a test configuration is loaded onto the FPGA only one time, and shifted to the other CLBs under the control of an external test unit, or automatically by the FPGA itself.

2.6.3.3 Fault Diagnosis. Fault diagnosis is an extension of fault detection, and most of the proposed techniques are extensions of the techniques from the previous section. In fault detection, a large number of devices could be tested in parallel with a single flag to report the error in any of the devices. Fault diagnosis requires identification of the specific location of the fault and its impact on overall FPGA function.

As with fault detection, most work has addressed diagnosis of faults in CLBs and the interconnect network. Most of the techniques use the programmability of the resources rather than proposing changes to the architecture to support test.

The BIST approach was extended by configuring alternate rows as blocks under test and output response analyzers [SLA97]. A second configuration swaps the arrangement so one row is the output response analyzer and the next is the block under test. All of the CLBs in a row are tested in parallel, so after two configurations, all of the CLBs have been tested. When a fault is detected, the row of the fault is available, but not the column. At this point, a third configuration is loaded, rotating the test structure 90 degrees in the array to test the columns. The third and fourth configurations are identical to the first and second, but rotated to test the columns instead of the rows. Thus after four configurations, the row and column of a faulty CLB is uniquely determined.

The Roving Self-Test Areas (RSTAR) is an on-line test method performed during circuit operation, without disturbing the operation of the application circuit [ASH⁺99, ASSE00, AES01, ASE04]. Similar to the configuration shifting method [DI99], RSTARs moves the self-test area across the configurable array. *Fault latency*, the interval between the occurrence of a fault and its detection, is thus bounded by the interval required to test the entire FPGA.

The basic concept of the RSTARS technique is shown in Figure 2.37. A portion of the FPGA array is initially assigned to be the horizontal and vertical STARS blocks. During operation of the application circuit in the remaining blocks, the VSTAR and HSTAR test unused portions of the array. When testing is complete, the HSTAR and VSTAR are relocated to new locations. One complete scan of each is enough to test the entire FPGA. While only the HSTAR (or the VSTAR) is needed to scan the CLBs, scans in both directions are used to locate faults in the interconnect.

The test is controlled by an external processor that is assumed to be reliable, called the Test and Reconfiguration Controller (TREC). The TREC can be imple-

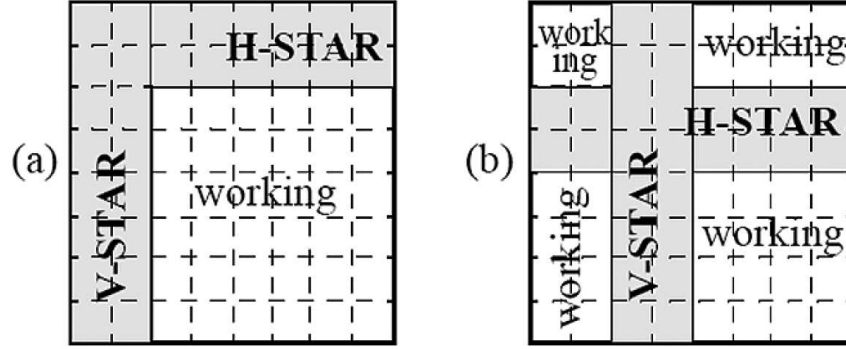


Figure 2.37: The Roving STARS technique uses Self Test areas that are relocated across the FPGA array during on-line testing [ASE04].

mented as an embedded processor on the FPGA, or externally using a separate ASIC. The relocation of the application circuit and STARS is done using pre-compiled partial configuration bit files controlled by the TREC. For a left-right sweep, the RSTARS approach needs $N/2$ swaps to move the columns across the array. To perform both a horizontal and vertical sweep across the entire FPGA, N swaps are needed. Runtime routing is not typically used to compute new configurations, although it can be done later if alternative configurations avoiding faulty resources are not available in pre-compiled form. This can be done while the main circuit continues operation, since STARS testing is done in unused areas of the chip [AES01].

Logic block testing in RSTARS is done by grouping six cells to uniquely determine the faulty CLBs (each cell is notional, and may actually larger than a single CLB). Six rotations of the configuration is sufficient to test all the cells. The concept is illustrated in Figure 2.38. In the figure, ‘T’ denotes a Test Pattern Generator, ‘O’ is the Output Response Analyzer, and ‘B’ is the Block Under Test.

Interconnect is tested in a similar manner, using different patterns of blocks, separated by some distance. A partially-exhaustive test pattern is used, but rotates on every test pass so the time required to perform a test is kept within bounds. Fault coverage is high after several passes, but fault latency is increased since it requires several passes before a test vector occurs that detects the fault.

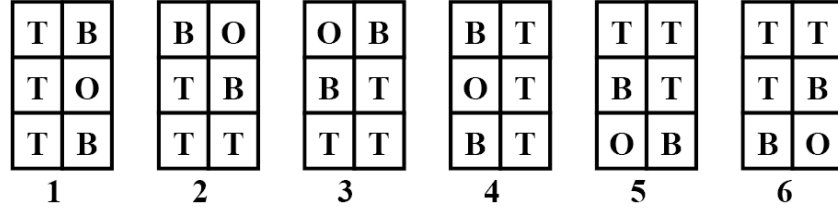


Figure 2.38: The Roving STARS technique uses six rotations test units to determine which CLB is faulty. ‘T’ denotes a Test Pattern Generator, ‘O’ is the Output Response Analyzer, and ‘B’ is the Block Under Test [ASSE00].

An additional RSTAR concept is the Partially Usable Block (PUB). RSTARS identifies the failure mode of the CLB, and what partial function it is capable of providing. For this to be successful, runtime routing is mandatory.

At the system level, the RSTARS approach proposes a three-tiered fault handling approach [AES01]. Following the location of a faulty resource, the TREC may:

1. Leave the STARS parked where they are, allowing the application circuit to continue operation in the rest of the array.
2. Apply precompiled or newly computed alternate configurations using spare CLBs and resources throughout the array.
3. When the spares are exhausted, de-allocate resources reserved for the RSTARS and use them in alternative configurations. This reduces later test capability, but allows graceful degradation.

The RSTARS system has been demonstrated in a limited manner on commercially available ORCA FPGAs. The test algorithm is effective at detecting and isolating faults, although the overhead incurred to perform the testing is significant. In addition, the requirement for an external control unit is a limitation of the system, as is the large number of alternate configuration bit files that must be stored. Without the pre-compiled bit files, routing around failed resources must be done at runtime. The RSTARS approach uses a slightly modified conventional router to avoid faulty

resources. Routing time is therefore likely to be significant. If devices fail frequently, routing may not complete before the next error occurs, causing system failure.

Finally, other methods also diagnose faults in FPGAs. Readback of the configuration memory can be used to localize the faults. TMR in the context of FPGAs was proposed by [DMP⁺98]. Fault diagnosis in I/O blocks was examined in [RWCG02].

III. Research Goals

This chapter presents the four goals of this research. The methodology for achieving these goals is addressed in the next chapter.

3.1 *Motivation*

Chapter II made several important observations that will shape future computer architectures:

- For silicon CMOS, Moore’s Law may no longer apply within the next several process generations.
- Both silicon CMOS and potential replacement technologies will be more difficult to fabricate reliably, and more likely to fail in operation.
- Fault tolerance is a viable way to use unreliable device technologies in commercial as well as space and military applications.
- Architectural fault tolerance can provide a lower cost alternative to fabricating devices with extremely low defect rates.
- Conventional fault tolerance involves temporal or spatial redundancy, which must be carefully balanced against available area and power in real devices.
- Methods need to be developed to combine fault tolerance, reconfigurable computing, and computer architecture technologies together to address this problem.

3.1.1 Four Goals. As a foundational effort in fault tolerant computer architecture at AFIT, this research defines a fault and defect tolerant computer, determines how it is different from a conventional computer, and identifies capabilities that must be present to achieve reliability goals. Relating this back to the device technologies, it determines the minimum reliability characteristics needed to compete with conventional CMOS. From this general goal, four explicit goals are defined:

Goal 1: Develop a system architecture for the FDT computer, propose a concept of operations (CONOPS), identify required capabilities.

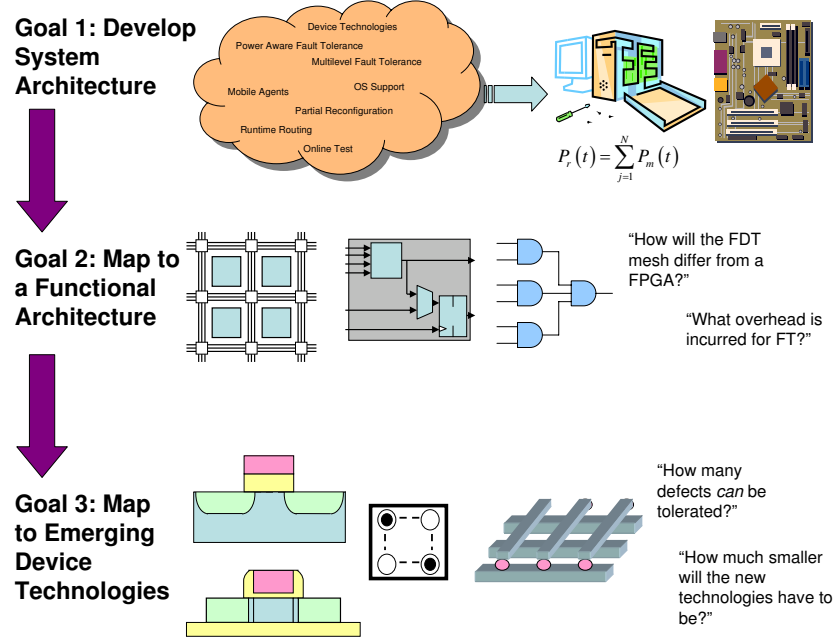


Figure 3.1: The three primary goals develop the FDT architecture from the system down to the device level.

Goal 2: Design a functional architecture and demonstrate it supports the functions identified in the previous goal.

Goal 3: Develop techniques to map the FDT architecture onto emerging technologies (e.g., molecular crossbars, quantum cellular automata (QCA), etc.) and characterize their reliability.

Goal 4: Extend the mathematical models for fault tolerance techniques.

The relationship between the first three goals is shown in Figure 3.1. Goal four is an enabling goal. The initial step in the research approach specifies a top-level architecture to determine required capabilities, and maps the architecture onto progressively lower levels. At each stage, fault tolerance effectiveness and overhead are examined. All four goals support the overall purpose: create a foundation for reliable computing using unreliable devices.

3.2 Goal 1: Develop the FDT System Architecture

“Develop the system architecture for the FDT computer, propose a concept of operations (CONOPS), identify required capabilities.”

3.2.1 Questions Addressed. The output of this goal is an architecture for a “fault and defect tolerant computer.” as well as addressing the following questions:

- What constitutes a fault and defect tolerant (FDT) computer?
- How is a FDT computer different from a conventional computer?
- What functions should a FDT computer be able to perform?
- How would a FDT computer operate?
- What fault and defect tolerance techniques must the FDT computer include?
- How will a FDT computer be compared against a conventional computer?
- If the FDT computer implements reconfiguration, how would it resemble a modern FPGA? How must it be different?

In general terms, a FDT computer is built from devices with a higher individual device defect rate and a higher operational failure rate than conventional silicon CMOS. Unlike a conventional processor, the FDT processor *requires* some level of fault and defect tolerance to achieve acceptable manufacturing yields and system reliability. Unlike modern fault tolerant systems which primarily target operational failures, FDT computers tolerate both manufacturing defects and operational failures. Operational failures in a FDT computer may occur more frequently than in current fault tolerant systems. Thus, the service provided by the FDT computer is reliable application programs execution in the presence of manufacturing defects, operational permanent faults, and operational soft errors.

The FDT computer can be compared to a conventional computer with several metrics, as defined in Chapter II:

- Defect Tolerance
 - Yield
 - Maximum Allowable Defect Probability (MADP)
- System Fault Tolerance
 - Availability
 - Reliability
 - Mean-Time-Before-Failure (MTBF)
 - Mean-Time-To-Repair (MTTR)
 - Maximum allowable Soft Error Rate (SER)
- Performance
 - Application speedup (or slowdown)
 - Maximum system clock speed
- Overhead
 - Amount of hardware redundancy
 - Die area
 - Amount of power increase

As the focus of this research is at the architecture rather than device level, limited information is available to develop performance and overhead estimates. Thus, defect tolerance is the primary metric used for goal one. Hardware overhead is also considered for goals two and three.

3.2.2 Quantifiable Goals. It is now possible to define quantifiable goals for the fault and defect tolerant computer architecture to be developed in this research.

Goal 1.1: Develop the system architecture for a FDT processor capable of meeting the following evaluation metrics:

- *Provide a manufacturing yield of $> 70\%$ for a process technology with an individual device defect probabilities greater than 10^{-6} for a representative microprocessor containing one million logic transistors and 100 million cache transistors.*
- *Provide architectural fault tolerance support for soft errors and single event upsets occurring in memory.*

Goal 1.2: Develop a FDT system concept of operation (CONOPS) and demonstrate it supports the performance criteria.

Goal 1.3: Develop mathematical models to demonstrate that the combined capabilities achieve desired performance criteria.

The choice of target values is intended to be representative of modern computer architectures and fabrication processes. The yield figure is chosen as a typical value for standard CMOS. Current CMOS processes have device defect rates typically less than 10^{-6} , thus 10^{-6} to 10^{-3} represents a range of emerging technologies with defect rates inferior to current fabrication limits. The value of 10^{-6} serves as a minimum threshold. The values for logic gates and cache transistors were chosen to represent a typical modern microprocessor.

3.3 Goal 2: Design the Functional Architecture

“Design a functional architecture and demonstrate it supports the capabilities identified in the system architecture.”

3.3.1 Questions Addressed. Goal two moves the system architecture developed in the first goal to a functional or logical level. Thus, a functional architecture capable of supporting the fault and defect tolerance techniques specified in the system architecture is developed. This goal addresses the following questions:

- How will the FDT processor architecture differ from a conventional microprocessor?
- How will the FDT processor operate?

- How will it be configured at startup?
- How will it reconfigured in operation?
- How will it be tested?
- What level of fault tolerance does the FDT processor provide against defects, operational permanent faults, soft errors, and SEUs?
- What is the overhead incurred compared ASIC implementations of a micro-processor?

The service provided by the functional architecture is the ability to implement a general purpose processor, while incorporating the fault tolerance techniques specified in goal one.

Performance metrics for the functional architecture of goal two are similar to those of the system level, although information is now available to develop estimates for performance and overhead. The following metrics are used:

- Defect Tolerance (i.e., yield, MADP)
- Overhead (e.g., hardware, die area, power)

Information on device technology is not available at this stage. Thus, estimates are based on logical gate counts and other methods that do not rely on device characteristics.

3.3.2 Quantifiable Goals. This goal has two parts: design the FDT processor; and develop models for its performance and overhead. In this manner, it will be possible to show that the FDT processor implements the functions required by the system architecture of goal one and meets the system reliability requirements.

Goal 2.1: Develop the architecture of the FDT processor such that it supports the capabilities identified in the system architecture.

Fault detection. The FDT processor should be able to detect and diagnose:

- Hard faults in the application and fault tolerance logic.
- Soft errors in the application logic.
- SEUs in the cache memory.

Fault masking. If fault masking is incorporated at the hardware level, the FDT processor should be able to provide the *fault coverage* specified in the system architecture (Goal 1).

Fault diagnosis. The FDT processor should be able to diagnose the location of a fault, down to the level of granularity of the smallest reconfigurable unit (e.g., down to the column for a column-wise reconfigurable mesh). While not an explicit focus of this research, test methodology should be discussed at a high level.

Fault isolation. As required by the system architecture, the FDT processor should be able to limit the impact of a fault on the overall system to some portion of the system.

Fault recovery. The FDT processor should have the following fault recovery capabilities:

- When a fault is detected, the FDT processor should support fault recovery to a known state and allow resumption of operation.
- If required by the system architecture, the FDT processor should support dynamic reconfiguration at the specified level of granularity.

Goal 2.2: Develop analytical estimators for overhead and performance of the FDT processor versus a conventional architecture. Develop estimators (independent of device technology) for hardware overhead and manufacturing yield.

3.4 Goal 3: Map the Functional Architecture onto Emerging Technologies

“Develop techniques to map the FDT architecture onto emerging technologies (e.g., molecular crossbars, quantum cellular automata (QCA), etc.).”

3.4.1 Questions Addressed. This goal examines the implementation of the FDT computer at the device level. The functional architecture created in the previous goal will likely be implemented as conventional digital logic. As such, it can be mapped to any device technology that implements boolean logic operations. The emerging device technologies examined in Chapter II have capabilities different from modern CMOS and may be able to implement some aspects of the FDT processor more effectively than CMOS. Thus, the overhead incurred by the fault tolerant architecture cannot be derived by simply examining the circuit at the digital logic level (i.e., by counting gates). An examination of the problem at the device level is needed to develop detailed comparisons between the technologies. The focus is thus not on whether or not the FDT processor can be implemented using one of the emerging device types, but rather how it would be implemented, what unique benefits it would obtain, and how it compares to a conventional CMOS implementation.

The research addresses the following questions:

- How would the fault tolerance techniques used in the FDT processor be implemented with non-CMOS technologies?
- How would hardware cost, yield, power, and speed be estimated using non-CMOS device technologies?
- Compared to modern CMOS, how much smaller/more power efficient/faster will the new device technologies have to be to overcome the overhead induced by the fault tolerant circuitry?
- Can the unique characteristics of the new technologies overcome some of the overhead?

- Based on current predictions, is it feasible to implement the FDT processor architecture? If not, can minimum limits be defined for the device technologies that must be exceeded to allow “real world” use of these devices?

This step limits examination to particular fault tolerance techniques, independent of the architecture developed in the first two goals. The service provided is the ability to implement the fault tolerance building blocks using one or more device technologies. At this level, it becomes possible to improve the estimators for performance and overhead first developed at the functional level. As such, the performance metrics at this stage are:

- Performance (e.g., speedup relative to CMOS), and
- Overhead (e.g., die area, power consumption).

3.4.2 Quantifiable Goals. The third goal has two parts: demonstrate that the emerging technologies can implement the FDT processor; and develop models for performance and overhead.

Goal 3.1: Demonstrate analytically how the fault tolerance techniques used in the FDT processor may be implemented using one or more of the following emerging device technologies:

- *Quantum Cellular Automata,*
- *Molecular Crossbars, or*
- *Nanoscale Silicon CMOS.*

Goal 3.2: Develop a methodology for estimating the FDT processor hardware area, power consumption, and operating speed when implemented using one or more of the aforementioned technologies.

Goal 3.3: Determine the minimum performance characteristics of a device technology necessary to fabricate a processor with the characteristics described in Goal 1. The minimum performance characteristics include:

- *Maximum Allowable Defect Probability (MADP),*
- *Size,*
- *Switching speed, and*
- *Power consumption.*

Quantum Cellular Automata is the target technology for this goal. As device technologies are still under development, the purpose of this goal is not to simulate or characterize the entire FDT processor. Rather, the aim is to demonstrate how the FDT processor implementation on the device technologies differs from conventional CMOS, and how the difference affects overhead and performance estimates.

The second part of goal three develops estimators for hardware area, power consumption, and operating speed used at the architectural level. First order estimates are developed that can be used to make design tradeoffs. This research creates estimators based upon device characteristics currently available.

The final part of goal three returns to the overall system architecture to determine whether construction of a reliable microprocessor may one day be feasible using these device technologies. Previous work in this area has largely been to demonstrate device operation. Researchers have acknowledged fabrication and reliability problems, observing that some level of fault tolerance will be necessary at the architectural level for reliable operation. For these technologies to be adopted, system level performance must equal or match that of modern silicon CMOS. In this goal, a MADP target will be established, below which the devices will not compete with conventional silicon CMOS.

3.5 Goal 4: Develop an accurate analytical model for NAND Multiplexing

“Develop an accurate analytical model for von Neumann Multiplexing at small and moderate levels of redundancy.”

3.5.1 Questions Addressed. This goal supports the first three goals by improving the model for a fault tolerance technique that could be used in the FDT processor architecture. NAND Multiplexing has not seen widespread use in current applications due to its requirement for large levels of redundant hardware. At the low defect rates common to modern processes, less aggressive fault tolerance techniques are sufficient. NAND Multiplexing is more effective than other techniques in the extreme defect ranges, from 10^{-5} to 10^{-2} . For technologies much smaller than silicon CMOS but much more defect prone, the overhead may be acceptable.

An approximation for the performance of NAND Multiplexing at large levels of redundancy was proposed by von Neumann in [vN56]. Since then, other models have been proposed for small and medium levels of redundancy. However, these models are incomplete and in some cases erroneous (cf., Chapter II). Due to the large number of devices in a microprocessor, even a small error in a yield estimate for a single device can become an unusable result at a larger scale. Thus, an accurate model is essential to determine how NAND Multiplexing can be used at the large scale.

The research addresses the following questions:

- Why is the model developed by [HJ02] incorrect, as claimed by [NPK04]?
- What is the actual analytical model?
- What use does NAND Multiplexing have in the FDT processor?

3.6 Research Contributions

Study of these problems will make the following contributions:

- A system architecture for a FDT computer, combining fault tolerance techniques at several levels of abstraction.
- Determine required performance properties of the FDT computer.

- Determine the partitioning of fault detection, diagnosis, and recovery tasks required to implement DT/FT functionality between modules on the chip, BIOS, and the operating system.
- Develop techniques to map the generic FDT processor onto various device technologies.
- Develop yield models for the FDT processor using FT/DT techniques.
- Develop an accurate analytical model for NAND Multiplexing, a basic fault tolerance techniques, at small and medium levels of redundancy.

3.7 Summary

This chapter establishes the goals of this research. Four goals are established, developing an architecture for a fault and defect tolerant computer from the top level architecture down to the device level. Quantifiable metrics are established to gauge success. The next chapter examines the methodology used to address these goals.

IV. Methodology

This chapter explains the methodology used to achieve the research goals. The major tasks in each goal are defined, task dependencies are highlighted, and the methodology for achieving the objectives are presented. In addition, scoping assumptions are given to bound the effort.

4.1 *Problem Scope*

This research spans several disciplines and areas of electrical engineering. The problem combines computer architecture and device technology, traditionally independent fields. Several assumptions and starting conditions are made to focus research:

Technology independence. As no device technology has emerged as the clear choice to replace silicon CMOS, this research is as independent of device technology as possible. The main research focus is at the architectural level. As developed in Chapter V, the yield and hardware cost models are based on devices (i.e., transistors) and do not explicitly model wires and other structures. These techniques are extended to a particular device technology in Chapter X, illustrating the changes that must be made to incorporate the key capabilities and limitations of a target technology.

Hardware Scope. Modern computers are made of several components connected on circuit boards. It is reasonable to assume that dual-technology systems will be built: a high performance non-silicon technology for the processor, while support hardware is implemented using CMOS microchips. Thus, the microprocessor is the focus of research. However, the techniques proposed are equally applicable to other large circuits.

Key Performance Criteria. While operational failures and soft errors will be a problem, the first challenge to overcome is manufacturing yield. The FDT processor developed in this research includes some functionality to detect and correct these types of faults, but the focus of analysis is on yield rather than reliability.

4.2 Goal 1: Develop the FDT System Architecture

The general approach to this goal is analytical. By examining and modelling various combinations of fault tolerance techniques, system yield models are developed. Trade-offs between the various fault tolerance techniques, implemented at different levels of the architecture, are investigated. From this information, the system architecture and CONOPS are defined, and fault tolerance techniques selected. The theoretical models show that the system yield requirements can be met.

Goal one is addressed in the high level architecture described in Chapter VII. Supporting yield and hardware cost models are developed in Chapter V.

4.2.1 Tasks. To meet the research objectives of goal one, several tasks are defined:

1. Propose initial system concept of operation (CONOPS).
2. Determine the appropriate set of fault tolerance techniques.
 - (a) Identify possible fault tolerance techniques.
 - (b) Develop analytical model for yield.
 - (c) Determine whether system meets performance goals.
 - (d) Adjust set of fault tolerance techniques and repeat Steps 2a-2d.
3. Develop the detailed system architecture implementing the identified fault tolerance techniques.
4. Develop criteria to compare FDT computers to each other and to conventional computers.
5. Analyze the effectiveness of the top level system.

The first step proposes an initial system level concept of operations for the FDT computer. As discussed in Chapter I, fault tolerance techniques can be incorporated into the system architecture at multiple levels, from the device level up to the operating system and application layers. The FDT processor proposed in Chapter VII

combines several fault tolerance techniques to achieve the system design objectives. This step examines the different techniques available, how the techniques might operate in support of each other, and determines an initial set to examine in detail.

The next step determines the set of fault tolerance techniques needed to achieve the performance goals. In some cases, analytical models for the techniques already exist, while others require development. When feasible, analytical models for system performance are created to model the multiple techniques used in concert. These models determine whether a system incorporating a given set of FT techniques will meet the goals. These models are introduced in Chapter V. An iterative process is used to determine the most effective set of FT techniques.

Once the required set of fault tolerance techniques has been identified, a system architecture is created to implement the techniques. At this stage, it is not necessary to develop functional or circuit level models of the architecture. Goals two and three examine the problems of implementing the architecture at lower levels. Instead, the product of this step consists of a concept of operations and a description of the required fault tolerance methods. The final step considers how the FDT design may be compared to a conventional computer architecture.

For Goal 1.1, yield is the primary quantifiable metric. Results for individual components are computed analytically and compared to Monte Carlo simulation results obtained with Matlab[®]. Analytical models for complicated architectures such as the overall cache memory or CPU become extremely complex due to multiple dependencies. For these situations, yield results are obtained using Matlab[®] simulation. Goals 1.2 and 1.3 use an analytical approach. It is sufficient to show the proposed system architecture is feasible to build, and can meet the design objectives of Goal 1.1.

4.2.2 Scope and Parameters. Analysis of the system is limited to the processor. It is assumed that the remainder of the system is constructed from reliable silicon CMOS components. Thus, the analysis evaluates the ability of the system to imple-

ment an application CPU reliably in the presence of defects and operational faults. The parameters of the system include initial fabrication device defect probability, operational permanent fault rate, and operational soft error rate.

4.3 Goal 2: Design the Functional Architecture

The general approach to goal two is through a combination of analysis and simulation. The functional architecture is designed, and analytical models are developed for hardware overhead and yield. Yield expressions at the module level are validated by comparing analytical results with simulation results from Matlab[®]. Results for the entire cache and processor architecture are obtained through Monte Carlo simulation in Matlab[®]. The results of goal two are presented in Chapters VIII and IX.

4.3.1 Tasks. To meet the research objectives of Goal 2, the following tasks are defined:

1. Identify required capabilities (from Goal 1.1)
2. Develop functional architecture.
3. Develop analytical models for yield and hardware cost.
4. Determine the yield of the cache and overall FDT processor.

The first step depends on the results of Goal 1. The system architecture from Goal 1 defines the functions the lower level architecture must implement. Once defined, the functional architecture is developed. This step develops a logic level, device independent, model of a FDT processor which shows the architecture supports the required fault tolerance capabilities and identifies differences between the proposed FDT architecture and a conventional microprocessor. Device independent yield and hardware cost estimates are used to compare the FDT architecture to conventional, non-fault tolerant, architectures.

4.3.2 Evaluation. Monte Carlo simulation is the primary method used to evaluate the yield performance of the FDT processor. Having chosen a defect model and developed the functional model, simulation is used to show the required yield target can be achieved. Hardware cost model validation is done analytically from the bottom up, using primitive models to build more complicated structures up to the processor level. As discussed in the previous section, Goal 2.2 is evaluated analytically.

4.4 Goal 3: Map the Functional Architecture to Emerging Technologies

The approach to this goal is primarily analytical, since detailed information on potential device technologies is limited. The results supporting this goal are presented in Chapter X.

4.4.1 Tasks. To achieve the research objectives of Goal 3, the following tasks are defined:

1. Examine the capabilities of the emerging device technologies. QCA is selected as the target technology.
2. Develop or adapt simple logical operations and other structures using the target technology.
3. Determine how the yield and hardware cost models from the previous goals must be modified to apply to the target device technology.
4. Develop estimators for size, power, performance.
5. Establish the minimum size, speed, and defect probability characteristics for the emerging technologies to compete with silicon CMOS.

The first step examines the capabilities of the emerging device technologies and justifies the selection of QCA as the target. Starting from simple, Boolean logic gate building blocks, conceptual designs for the key fault tolerance techniques used in the FDT processor are developed. QCA Designer is used where appropriate to develop the physical layouts for fault tolerant circuits. During this process, the unique

characteristics of QCA that result in an implementation that differs from conventional CMOS are identified.

Finally, the last step in the research returns to the top level. The lessons learned throughout the research process as the system architecture was specified, developed at the digital level, and mapped to proposed technologies. Chapter XI addresses the “big picture” questions defined in Section 3.4.1. The performance and overhead models first developed at the system level, and expanded down to the device level, are used to determine the minimum device characteristics necessary to compete successfully with conventional silicon CMOS architectures.

4.4.2 Evaluation. Analysis is again the primary method of accomplishing this goal. Hardware layouts for the key fault tolerance techniques are developed to show function and form the basis for hardware cost estimation. Analytical yield models are constructed analytically, and compared using Matlab[®] and MathCad[®].

4.5 Goal 4: Develop an analytical model for NAND Multiplexing

The approach to this goal is analytical. The mathematics resulting from this analysis are in Chapter VI. The mathematical model is validated using Matlab[®] simulation and the PRISM tool [NPK04]. Confidence intervals are computing using the method described in Chapter VI.

V. Tools and Models

This chapter introduces yield and hardware cost models for the fault tolerance methods used in the FDT computer. Several common module-level techniques such as R-modular redundancy and modular reconfiguration are illustrated. A new technique, TMR-protected reconfiguration, is proposed. TMR-R combines the benefits of TMR and modular reconfiguration and is be used extensively in later chapters. Mathematical models for common memory fault tolerance techniques are shown, including error correcting codes and spare rows/columns.

In addition to yield, models for hardware cost are introduced. The hardware cost model is intended for yield modelling rather than area estimation, but does prove useful in estimating hardware overhead. The hardware cost models are introduced in Section 5.5. NAND multiplexing, a fault tolerance technique, is described only briefly. A detailed mathematical model is developed in Chapter VI.

5.1 *Yield Models*

5.1.1 Basic Yield Models. Summary works of yield modelling are found in [Kor89, KK98]. These models create analytical probability models of the distribution of defects on the wafer, and of the impact of defects on overall device function. Let X be a random variable denoting the number of faults in the chip. For a circuit with no fault tolerance, the yield is simply the probability that no defects occur. Thus, chip yield, Y_{chip} , is simply

$$Y_{chip} = P(X = 0). \quad (5.1)$$

For circuits with fault tolerance capabilities, the circuit can operate correctly when $X > 0$. More complicated yield models have been developed for these situations.

5.1.2 Clustering. Various models have been developed to account for a non-uniform distribution of defects on a wafer and the resulting impact on yield. The most common statistical models are the Poisson model and the Negative Binomial model.

The Poisson model approximates a binomial distribution for large N and small p . The Poisson model assumes independence between defects resulting in an unclustered distribution of defects on the wafer. In real fabrication processes, however, defects tend to cluster together. Many defect distributions have been proposed, including gamma, triangle, delta, and exponential distributions, and are summarized in [MVM90]. The Gamma distribution is widely used, and has been shown to be a good fit to real world data [Cun90]. Averaging of the Poisson yield expression over the range of values for the number of defects per chip, λ , distributed according to the Gamma distribution, leads to the negative binomial distribution for chip yield.

Using the Poisson distribution,

$$P(X = k) = \frac{e^{-\lambda} \lambda^k}{k!}, \quad k \geq 0, \quad (5.2)$$

and the chip yield is

$$Y_{chip} = P(X = 0) = e^{-\lambda}. \quad (5.3)$$

For a chip composed of multiple modules,

$$P(X = k) = \frac{e^{-N\lambda} (N\lambda)^k}{k!}, \quad k \geq 0, \quad (5.4)$$

where N is the number of modules in the chip, and λ is the number of defects per module (or per device, depending on the level of abstraction). Yield in this form is simply

$$Y_{chip} = P(X = 0) = e^{-N\lambda}. \quad (5.5)$$

To derive the yield expression using clustered defects, a compounding procedure is used. Compounding considers λ as a random variable rather than a constant. Let l be this defect rate. Starting from the expression for unclustered (Poisson) yield as a

function of the defect rate (5.2), the defect distribution of the chip with clustering is

$$P(X = k) = \int_0^{\infty} P(X = k|l) \cdot f_L(l) dl \quad (5.6)$$

where $f_L(l)$ is the *compounder* or *mixing function* [KK98]. The Gamma distribution uses two parameters, λ and α , and

$$f_L(l) = \frac{\alpha^\alpha}{\lambda^\alpha \Gamma(\alpha)} \cdot \lambda^{\alpha-1} \cdot e^{-\alpha l/\lambda}. \quad (5.7)$$

The clustering parameter, α typically ranges from 0.3 to 10. As $\alpha \rightarrow \infty$, the distribution approximates a Poisson distribution. Evaluating (5.6) using this distribution yields the negative binomial distribution for the number of defects in the chip,

$$P(X = k) = \frac{\Gamma(\alpha + k)}{k! \Gamma(\alpha)} \cdot \frac{(\lambda/\alpha)^k}{(1 + \lambda/\alpha)^{\alpha+k}}. \quad (5.8)$$

Thus, the yield of a chip with no fault tolerance using a clustered defect model is

$$Y_{chip} = P(X = 0) = (1 + \lambda/\alpha)^{-\alpha}. \quad (5.9)$$

For chips with multiple modules, the clustered yield is

$$Y_{chip} = (1 + N\lambda_1/\alpha)^{-\alpha}, \quad (5.10)$$

where λ_1 is the probability of failure of a single device (i.e., transistor), and N is the number of modules in the chip.

5.1.3 *Multiple Components.* For the Poisson case, the yield of a chip containing multiple independent components is found by simply multiplying the yields of the individual components.

$$Y_{chip} = \prod_i Y_i. \quad (5.11)$$

For the case of clustered defects, the compounding procedure from (5.6) is used, but care must be taken to perform a single compounding step for the entire chip rather than separate compounding steps for each module, as the clustering of faults in one module is not independent of clustering in the other modules [Nik96, Sta93, NV99]. Therefore, a single compounding step is performed using the average number of faults in the complete chip, or

$$\lambda_{chip} = \sum_i \lambda_i. \quad (5.12)$$

To simplify the integration, which contains different λ values for the different modules, scaling constants are used, or

$$\delta_i = \frac{\lambda_i}{\lambda_{chip}}, \quad (5.13)$$

where δ_i is the probability of observing a fault due to component i .

In practice, the integrals developed using compounding for multiple modules are difficult to solve analytically and numerical integration is commonly used. In many cases, these integrals involve multiplying very large numbers of elements by very small probabilities. Accuracy is limited to the precision of the floating point representations used in computer software. Extended precision floating point libraries can be used to increase the accuracy of analytical results [LL05, Var05]. More commonly, Monte Carlo simulations of the memories are used to estimate yield.

5.2 Fault Models

Memory yield models sometimes include multiple types of faults. Some memory models only consider memory cell faults that disable a single memory bit in the array. Other models add multiple memory bit failures (i.e., a single fault disables two or more memory bits), as well as row and column faults (i.e., a single fault disables an entire row or column of memory cells). Herein, faults are modelled at the single-transistor level. In a typical memory cell, a single transistor fault disables the memory cell. If the fault occurs in a row or column decoder, it disables the entire row or column.

For combinational logic, a variety of fault models have been proposed. Von Neumann faults [vN56] invert the logical state of a logic gate. Stuck-at zero (one) faults force the output of the gate to a logic low (high) value. Some models incorporate parametric faults that change the timing or current flow characteristics of the circuit. Any faults in combinational logic are assumed to disable that module.

At the device level, faults can occur in the transistors or interconnects. In many yield models, interconnect faults are combined with nearby devices and modelled together. Thus, device defect probability, λ_1 , includes both events: device failure and associated interconnect, or,

$$P(\text{Combined device failure}) = \lambda_1 = P(\text{device fails} \vee \text{interconnect fails}). \quad (5.14)$$

This model is most appropriate for device technologies in which the probability of transistor (or switch) failure is much greater than interconnect failure. For other technologies, interconnect faults cannot be assumed to be distributed evenly among devices. Thus, wiring must be modelled as well and more complicated models are required. An example of how the models change for one such technology, quantum cellular automata (QCA), is shown in Chapter X. For the FDT processor, described in Chapters VII, VIII, and IX, the device-centric model is used instead.

5.3 Key Fault Tolerance Techniques

Reliability is one of several competing requirements in cache design. In a typical cache, performance is the key design criteria. Capacity is another key goal. These two factors are often at odds with the redundancy-based techniques used for fault tolerance. Extra hardware is used to provide spare rows or columns as well as to provide error detection and correction capabilities reduces the capacity of the cache. In addition, increased propagation delay due to path length increases latency. Therefore, the goal of fault tolerant cache design is to achieve acceptable manufacturing yields and operational reliability with the least amount of redundant hardware. For this reason, a variety of techniques have been developed to model defects and the performance of fault tolerance techniques. This section summarizes several of the key techniques.

5.3.1 NAND Multiplexing. von Neumann Multiplexing (VNM), also known as NAND Multiplexing, was first proposed in [vN56]. Analytical models for NAND Multiplexing performance at low levels of redundancy were developed in [HJ02]. After identifying flaws with the initial analytical model, statistical simulation was used to estimate performance in [NPK04,BS04b]. This research has derived the first accurate model for the performance of NAND multiplexing at moderate levels of redundancy. This detailed model is described in Chapter VI. A similar technique for three-input majority gates, MAJ-3 Multiplexing, is examined in [RB05].

Multiplexing can be more effective than RMR for applications in which large amounts of redundancy can be supported. Multiplexing replicates an operation in both a parallel and a serial manner. Signals are replicated to create bundles of N parallel signals. Operations are repeated in M stages: one *executive stage* followed by one or more *restorative units* [vN56]. As the number of operations in parallel, N , is increased, or the number of stages used, M , is increased, the probability of correct output improves.

Multiplexing has limited practical application in situations where the entire logic chain cannot be replicated in parallel N times. For example, it is not feasible to replicate the bit or word lines to a memory cell N times. Thus, the N output lines of a module protected by multiplexing are usually reduced to a single line using a majority gate. Oftentimes, the reliability of this majority gate limits the benefit of NAND Multiplexing or MAJ-3 Multiplexing in a manner similar to RMR. Thus, for NAND Multiplexing,

$$Y_{VNM} = Y_{VNm od} \cdot Y_{majgate}, \quad (5.15)$$

where $Y_{majgate}$ is from (5.5), with the appropriate value for $N_{majgate}$.

Thus, NAND Multiplexing is very effective when large amounts of redundancy can be used (i.e., more than 100 fold). NAND Multiplexing may be necessary for device technologies with defect rates greater than 10^{-5} . However, as was demonstrated in this research, other fault tolerance techniques requiring fewer resources are sufficient in the range of $10^{-9} < \lambda_1 < 10^{-5}$. To compete with silicon CMOS, redundancy requirements must be kept as low as possible. For this reason, RMR and reconfiguration are used in preference to multiplexing in the FDT processor architecture proposed herein.

5.3.2 R-Modular Redundancy. R-modular redundancy is widely used and replicates the logic module R times [SNF04]. For the most common method, *Triple Modular Redundancy*, $R = 3$. A majority voter compares the outputs of the modules and outputs the most common value. For RMR to work correctly, at least $(R + 1)/2$ of the modules must function correctly. In addition, the majority voter must function as well. The analytical expression for yield, not accounting for clustering, is

$$Y_{RMR} = P_{majgate} \cdot P_{majmods} \quad (5.16)$$

where $P_{majgate}$ is the probability the majority gate functions. Unless device level reliability improvements are possible in the device technology, $P_{majgate}$ has the same λ_1 as other modules, and is modelled by (5.5) or (5.10) with N equal to the number of devices in the majority gate. A survey of majority gate designs is found in [BQA03]. $P_{majmods}$ is the probability the majority (i.e., at least $(R + 1)/2$) of the modules function, and is

$$P_{majmods} = \sum_{i=\lceil \frac{R}{2} \rceil}^R \left(\binom{R}{i} P_{\text{mod}}^i (1 - P_{\text{mod}})^{R-i} \right). \quad (5.17)$$

5.3.3 Modular Reconfiguration. Reconfiguration assumes the logic module can be implemented in more than one location on the chip. Testing determines a fault-free location to implement the module. In programmable logic devices, the application logic module can be implemented once, in a location chosen from the set of functional configurable logic modules. Reconfiguration can also be done in a fixed circuit by implementing R instances of the application logic module. Testing determines which of the R instances is functional. One of these functional modules is selected to connect to the rest of the circuit.

While flexible, programmable logic devices require a large amount of overhead due to redundant interconnections, configuration registers, and other circuitry. For a high-speed processor, the switched-module approach provides some fault tolerance with overhead similar to RMR. An example of modular reconfiguration is shown in Figure 5.1. The probability an application module using switched-module reconfiguration is correctable is

$$P_{reconf} = P_{switch} \cdot P(M), \quad (5.18)$$

where P_{switch} is the probability the switching circuit functions correctly and $P(M)$ is the probability at least one of the R modules functions, or

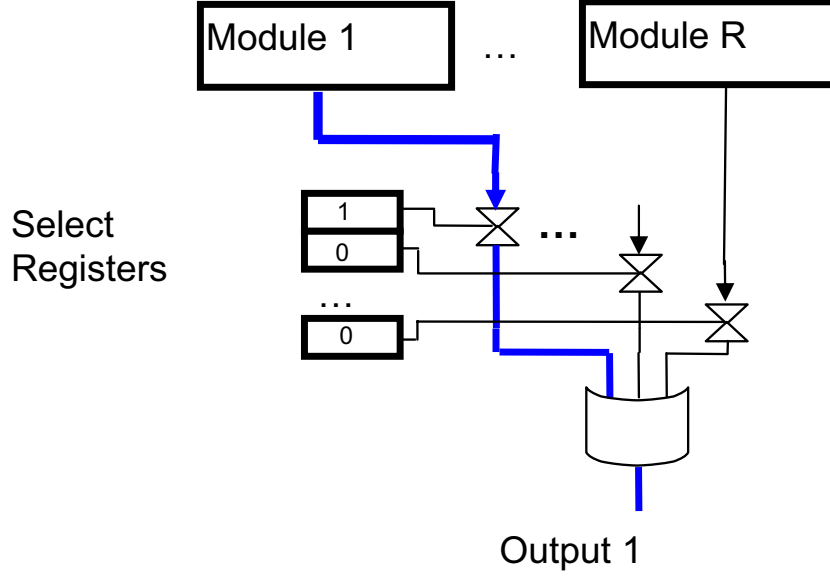


Figure 5.1: Modular reconfiguration has minimal overhead requirements to implement interconnect and switching. It is a very effective method of fault tolerance for moderate defect rates.

$$P(M) = 1 - P(\overline{M}) = 1 - (1 - P_{mod})^R. \quad (5.19)$$

It is also possible to combine reconfiguration with RMR by using reconfiguration to select R functional modules from a group of $R + S$ modules and passing the results to a majority voter. The probability this module is correctable is

$$P_{RMReconf} = P_{majgate} \cdot P_{majconnected}, \quad (5.20)$$

where $P_{majgate}$ is the probability the majority gate functions, and $P_{majconnected}$ is the probability the majority of the R connected modules function.

5.3.4 TMR-Protected Reconfiguration. Finally, RMR can be combined with module reconfiguration. Here, R modules are implemented of which T are connected to a majority gate. This technique provides additional protection against soft er-

rors. Herein, triple modular redundancy (TMR) protected reconfiguration (TMR-R) is used. An example of TMR-R is shown in Figure 5.2.

The yield of a TMR-R module is

$$Y_{TMRR} = P_{switchworks} \cdot P_{atleast2modswork}, \quad (5.21)$$

where $P_{switchworks}$ is the yield of the 3-input majority gate and the input selectors, and

$$P_{atleast2modswork} = \sum_{k=2}^R \binom{R}{k} Y_{mod}^k (1 - Y_{mod})^{R-k}. \quad (5.22)$$

To reduce power consumption, unused modules should be disconnected from power sources.

5.3.5 Threshold Gate Logic. Threshold logic gate (TLG) circuits have received some attention for fault tolerance and neural networks applications [LC67, Rei00]. Theoretically, TLG circuits can be made arbitrarily fault-tolerant using small to moderate amounts of redundant hardware while Boolean circuits cannot [Rei00]. Threshold logic gates can implement any Boolean function and could replace conventional Boolean gates. However, threshold logic design differs greatly from Boolean design, and new design and synthesis tools will be required [BQA03]. For the near to mid-term, computers will continue to be constructed from Boolean logic gates.

5.4 Memory Array Fault Tolerance

Several fault tolerance techniques used in computer memories are discussed in this section. They will be used in the design of the FDT cache in Chapter VIII.

5.4.1 Error Correcting Codes. Forward Error Correction is often used in memories where soft errors and Single Event Upsets (SEUs) occur due to radiation or electrical noise. ECC can correct errors induced through both transient events as well as manufacturing defects. The most common approach uses simple parity bits

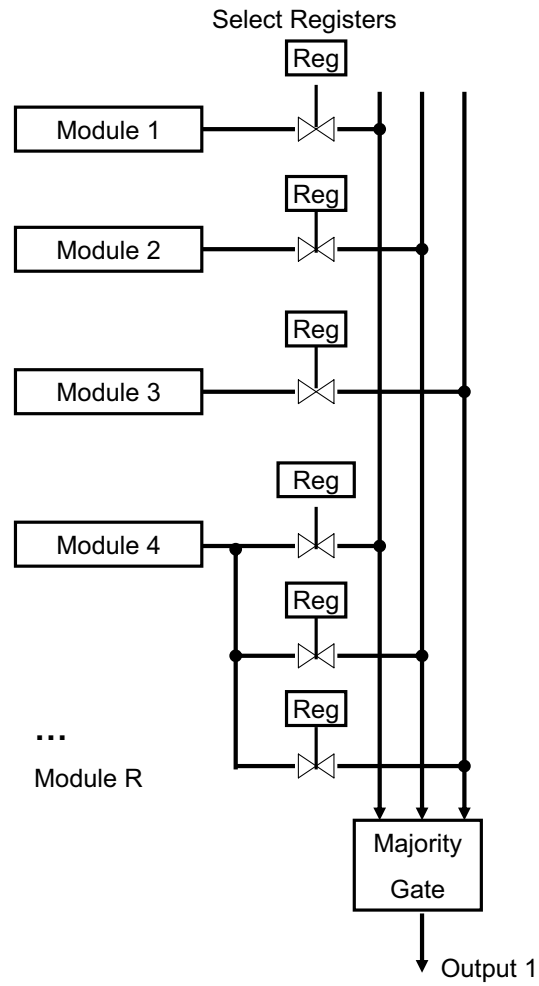


Figure 5.2: TMR-protected modular reconfiguration combines the benefits of reconfiguration with the soft error protection of TMR.

or Hamming codes. Modern microprocessors from AMD and Intel both use forms of Error Correcting Codes (ECC) in their cache to detect and correct single bit errors with minimal impact on operation. One simple method to model the performance of ECC on yield is

$$Y_{ECC} = P_{eccw}^W \quad (5.23)$$

where W is the number of ECC words in the cache, and P_{eccw} is the probability that a single ECC word contains a correctable number of errors. This probability is

$$P_{eccw} = \sum_{i=c-t}^c \binom{c}{i} p_{bit}^i (1 - p_{bit})^{c-i} \quad (5.24)$$

where c is the number of bits in each ECC code word, p_{bit} is the probability each memory bit will be functional, and t is the error correcting capability of the code used. In most simple ECC schemes, $t = 1$. In the simplest parity codes, $t = 0$, meaning the code can detect a single bit error in the code word, but cannot isolate the error location. In this case, the processor must recover from the error in a different way (e.g., hardware exception). For $t = 1$, the code can isolate the location of the error and correct it.

More complicated codes such as the extended Golay and Bose-Chadhuri-Hocquenghem (BCH) codes [Skl01,LDJC83], provide better detection/correction performance at the cost of increased hardware complexity and increased latency. Access time is usually a key concern for cache design and codes that can be decoded quickly are desirable. Parallelized implementations of some encoders/decoders have been implemented for extended Golay and other codes [BMH00,LDJC83], but add significant amounts of additional hardware. While the reliability of the memory array is improved, the reliability of the encoder/decoder module actually decreases due to the increase in the number of devices that can fail. Thus, there must be a balance of code complexity versus performance for a particular design.

5.4.2 Global Spares/Content Addressable Memories. The simplest memory fault tolerance approach uses global spares. In this case, a number of spare bits are available to replace faulty bits, with little or no restriction on their placement or use. In this idealized form, the yield of a chip can be modelled as

$$Y_{GS} = \sum_{i=b}^{b+s} \binom{b+s}{i} p_{bit}^i (1 - p_{bit})^{b+s-i}, \quad (5.25)$$

where b is the number of bits in the cache, s is the number of spare bits, and p_{bit} is the probability a memory bit is functional, or

$$p_{bit} = e^{-N_b \lambda_1}, \quad (5.26)$$

where N_b is the number of devices in a memory bit (i.e., cell). $N_b = 8$ in a typical dual read port memory architecture. The value for λ_1 is the mean number of defects *per transistor*.

It is usually not practical to implement spares with no limitations on their use. Restrictions on interconnect, fan-in, and fan-out typically limit spares to certain sections of the memory. The closest practical implementation of global spares is the Content-Addressable Memory [Lo93,Lo94]. In this approach, each memory cell stores the memory address in addition to the data bit(s). With each memory access, all CAM cells compare their stored address to the input address. The address will match for a single CAM cell. This cell performs the desired read or write operation. CAM architectures can suffer from slower speeds due to large fan-ins depending on the device technology. These problems may not be as significant for non-CMOS device technologies of the future.

5.4.3 Spare Rows and Columns. The most common memory fault tolerance technique uses spare rows and/or columns in the memory array. Post-manufacturing testing determines the locations of faults, and spare rows/columns permanently re-

place faulty elements. This can be done through laser fusing, or dynamically through registers. The yield equations for a memory containing spare rows (or spare columns), is similar to the equation for global spares,

$$Y_{SR} = \sum_{i=r}^{r+sr} \binom{r+sr}{i} p_{row}^i (1 - p_{row})^{r+sr-i} \quad (5.27)$$

where r is the number of required rows, sr is the number of spare rows available (without assuming they are functional a priori), and p_{row} is the probability a row is functional. This probability is

$$p_{row} = e^{N_b \lambda_1 N_r}, \quad (5.28)$$

where N_b is the number of devices per memory bit, λ_1 is the mean number of defects per device, and N_r is the number of memory bits per row.

Some architectures use both spare rows and spare columns. Closed form analytical expressions for this approach have not been found, but several approximations have been proposed. In [KK97], st replaces sr in (5.27), where st is the sum of the spare rows and columns. Another approximation, uses $st = sr \cdot sc$ [CPL⁺03]. In practice, statistical simulation is commonly used to evaluate the performance of these architectures.

5.5 *Hardware Cost Models*

Hardware cost models are used primarily to provide input information to the yield models. The hardware cost models are also used to estimate relative “cost” of fault tolerance hardware in terms of devices or chip area. A similar method of transistor counting is used to develop a model of the MIPS 32 bit RISC microprocessor [MP00]. Starting from primitives, more complicated structures are created. From these models, it is possible to estimate the number of transistors in the entire proces-

sor. This technique is used directly to estimate the number of devices in the non fault tolerant processor and improved FDT processor in Chapter IX.

Area estimation is more complex as it must consider interconnect lines. The area occupied by a circuit depends greatly on layout, the number of signals, and the distances that must be crossed. In addition, interconnect estimation is strongly dependent on device technology. For example, modern silicon CMOS uses several layers of vertically separated interconnect lines, connected by vias. Increasing the number of layers reduces the overall area. In other technologies, multiple layers are not possible. For these reasons, cost comparisons at the architectural level are performed at the logical level (i.e., independently of device technology).

In Chapter X, the models are mapped onto a specific device technology, quantum cellular automata. Here, the device counting models are extended to include the interconnect. This has significant effects on both yield models and hardware area estimation.

5.5.1 Primitives. The hardware model used in the next several chapters is based on silicon CMOS. For example, an inverter requires two transistors; a two-input NAND gate requires four. Table 5.1 summarizes the primitives used to construct larger circuits.

From the primitives, several larger circuits are used later. First, the $A - to - 2^A$ *Address Decoder with Enable Input* has a cost of

$$C_{dec} = 2^A \cdot C_{NAND}(A + 1) + (A + 1) \cdot C_{NOT}, \quad (5.29)$$

where A is the number of bits in the address. The decoder is used in the non fault tolerant cache design.

Table 5.1: Hardware cost primitives [Wak90, Man88]. The default unit is the transistor.

Module	Symbol	HW Cost
Inverter	C_{NOT}	2
NAND2	C_{NAND2}	4
NOR2	C_{NOR2}	4
x-input NAND	$C_{NAND}(x)$	2x
x-input NOR	$C_{NOR}(x)$	2x
Buffer	C_{buffer}	4
XOR2	C_{XOR2}	16
Transmission Gate	C_{tgate}	2
SR Latch	$C_{SRLatch}$	8
D Flip Flop, Pos. Edge Triggered	C_{dff}	22

Multiplexers are used in many circuits. The design of the X -to-1 by W_{out} *multiplexer* is from [Wak90]. The hardware cost is

$$C_{mux} = \log_2(X) \cdot C_{NOT} + W_{out} \cdot (C_{NAND}(X) + X \cdot C_{NAND}(1 + \log_2(X))), \quad (5.30)$$

where X is the number of inputs to select from, and W_{out} is the number of bits selected in parallel. This is useful when busses are used, as in the 32 bit processor design in Chapter IX.

Similarly, the design of the 1-to- X by W_{out} *demultiplexer with Enable E* is from [Man88]. The hardware cost expression is

$$C_{demux} = W_{out} \cdot X \cdot C_{NAND}(\log_2(X) + E) + \log_2(X) \cdot C_{NOT}, \quad (5.31)$$

where X is the number of outputs, and W_{out} is the width of the bus. $E = 1$ if an enable input is required.

Memory elements are used throughout the design. The W_{out} *bit register with common enable line* is also from [Man88]. Its cost is

$$C_{reg_opt}(W_{out}) = W_{out} \cdot (C_{dff} + 3C_{NAND2}) + C_{NOT}, \quad (5.32)$$

where W_{out} is the number of bits to be stored. This version uses a single enable line for all bits. Sometimes it is desirable to enable each bit individually. In this case, the hardware cost becomes

$$C_{reg_nonopt}(W_{out}) = W_{out} \cdot (C_{dff} + 3C_{NAND2} + C_{NOT}). \quad (5.33)$$

Majority gates are a basic element in both RMR and TMR-R. The R -input *Majority gate* (i.e., MAJ-R) design is from [HPS75]. A variation of this type of gate is commonly found in the *mirror adder*. Its hardware cost is

$$C_{majgate} = 4W_{out} \cdot \left(\frac{R}{\frac{R+1}{2}} \right), \quad (5.34)$$

where R is the number of inputs, and W_{out} is the number of output bits in parallel.

5.5.2 Fault Tolerance Circuits. The hardware cost of a circuit protected with R-modular Redundancy is

$$C_{RMR} = W_{out}C_{majgate}(R) + R \cdot N_{mod}, \quad (5.35)$$

where W_{out} is the number of bits in the module's output, R is the number of redundant modules, and N_{mod} is the number of devices in the module to be protected.

The hardware cost for a circuit protected with Modular Reconfiguration is

$$C_{reconf} = (W_{out}C_{NOR}(R) + C_{reg_opt}(R) + W_{out}R C_{tgate}) + R \cdot N_{mod}, \quad (5.36)$$

where W_{out} is the number of bits in the module output, R is the number of redundant modules, and N_{mod} is the number of devices in the module to be protected.

The hardware cost of a circuit protected with TMR-protected reconfiguration is

$$C_{TMR} = (W_{out}C_{majgate}(3) + C_{reg_opt}(3R-6) + (3R-6)W_{out}C_{tgate}) + R \cdot N_{mod}, \quad (5.37)$$

where W_{out} is the number of bits in the module output, R is the number of redundant modules, and N_{mod} is the number of devices in the module to be protected.

5.6 Summary

This chapter develops yield and hardware cost models for the fault tolerance techniques used in the fault and defect tolerant processor. Analytical models for CMOS yield prediction are adapted for RMR, modular reconfiguration, and TMR-R. In addition, the most common memory fault tolerance techniques, ECC, global sparing, and row/column sparing are summarized. TMR-R, a new fault tolerance technique combining the benefits of modular reconfiguration with the soft error protection provided by TMR, is introduced. A hardware cost model is introduced and yield expressions are developed for the key fault tolerance techniques. These models are used later to predict the yield and overhead requirements of the FDT processor.

VI. von Neumann Multiplexing

This chapter develops the first exact analytical model for the performance of von Neumann (NAND) Multiplexing, an important fault tolerance technique first created in 1956 [vN56]. To date, all of the models for the effectiveness of this technique have been approximations. An analytical model was recently proposed by Han and Jonker [HJ02], but found to produce inaccurate results. This model has been examined, the errors found, and an improved model produced to correctly predict the performance of this technique. MATLAB simulations have verified the correctness of this improved model. Thus, researchers now have the first accurate performance model for this fault tolerance technique. This result supports Goal 4, providing tools and models to support fault tolerant system development.

6.1 *Introduction*

The miniaturization of silicon CMOS transistors has advanced roughly in step with Moore's Law for forty years. Device sizes continue to shrink as new solutions are found to fabrication problems. The end of Moore's Law has been foretold for almost as long, as physics challenges become increasingly difficult to overcome. Since transistors are made from finite numbers of atoms, there is a limit to the minimum size of a conventional MOS transistor. Indeed, that limit is perhaps only a decade away. Already quantum tunnelling effects are beginning to significantly impact device operation, increasing leakage currents, as well as making it more difficult for the gate to control the flow of current in the transistor.

For some time, alternative technologies have been examined as a replacement for silicon CMOS. In addition to hybrid silicon devices such as dual-gate and vertical gate transistors, new device types such as single-electron transistors and molecular crossbars have been proposed. Although none of these devices has yet emerged as the successor for conventional CMOS, they have several things in common. As a penalty for small size, the devices are more difficult to fabricate, more subject to manufacturing defects, and more likely to suffer from failures during operation.

To make use of these devices, the underlying architecture must be able to detect and tolerate errors while producing correct results. A variety of techniques have been proposed to do this, typically involving the incorporation of redundant devices into the circuit, or the use of reconfigurable logic to move the application circuit away from defective devices. One such redundancy technique, *NAND Multiplexing* was first proposed by von Neumann [vN56], and modelled for large amounts of redundancy. An analytical model for smaller amounts of redundancy was proposed by Han and Jonker [HJ02,HJ03]. Norman et al. [NPK04,BS04b] observed that this model does not account for dependence between the redundant inputs, and proposed a probabilistic model checker-based approach to model performance. A second analytical model was proposed early in [SNF04], but set aside in favor of the binomial model from [HJ02].

This chapter presents the first exact analytical model for the the performance of von Neumann Multiplexing at moderate levels of redundancy. The new combinatoric model accounts for dependence between the inputs to the NAND gates. This new model is somewhat similar to a recent combinatoric model presented for Three-Input Majority Multiplexing [RB04,RB05]. However, it appears that one type of error was omitted in [RB05], that is accounted for in the model in this chapter. In addition, three additional types of faults are modelled: output stuck-at-zero, output stuck-at-one, and input stuck-at-zero. The results of the combinatorics model are compared with results obtained via MATLAB simulation as well as the probabilistic model checker PRISM [NPK04,PNK04,BS04b,BS04a]. The improved model in this chapter model is up to 20% more accurate than all known previous analytical models and the PRISM simulator, and produces results several times faster than PRISM.

6.2 *von Neumann Multiplexing*

6.2.1 Overview. In early computers, logical functions were realized using vacuum tubes. These devices were prone to failure, and the mean time before failure of a computer constructed of vacuum tubes was quite low. Research began in the area of fault tolerance and, in 1952, John von Neumann investigated the possibility of per-

forming reliable operations with unreliable components through redundancy [vN56]. Two methods were investigated, *majority voting* and *multiplexing*. Both methods use a group of logical gates (any of which can fail) in place of a single unreliable gate. Von Neumann showed that if the probability each gate fails is sufficiently small, and the errors in each gate are independent, a high probability of a correct result can be achieved.

6.2.1.1 The NAND Multiplexing Unit. The purpose of the NAND Multiplexer is to reliably perform the boolean NAND operation in the presence of errors that change the operation of the device. A ‘von Neumann fault’ [vN56] inverts the correct output of a NAND gate. The NAND Multiplexer circuit performs the NAND operation redundantly (see Figure 6.1), increasing the probability of correct output over that of a single NAND gate.

In the Multiplexing technique, logic signals are represented by bundles of signals. For example, a NAND gate may have two inputs, X and Y , and one output, Z . Each signal is represented by a bundle of N signals. If there are no errors in a signal, all N lines in the bundle have the same value. If errors are present, some fraction of the lines have the opposite value. A threshold, $\Delta \in (0, 0.5)$ is defined such that when no more than ΔN of the lines in the bundle are stimulated (i.e., logic ‘true’ or ‘1’), the logical value of the variable represented by the bundle is interpreted to be ‘false’ or ‘0’. Likewise, at least $(1 - \Delta)N$ lines must be asserted for the logical value of the variable represented to be considered ‘true’ or ‘1’. If the number of asserted lines in the bundle is between these two thresholds $(\Delta N, (1 - \Delta)N)$, the state is undecided, and a malfunction is assumed.

The NAND Multiplexer is composed of two parts: the *Executive Stage* and one or more *Restorative Stages*. Each restorative stage is nothing more than two executive stages connected in series. In most cases, adding more restorative stages or increasing the bundle size N makes the NAND operation more reliable.

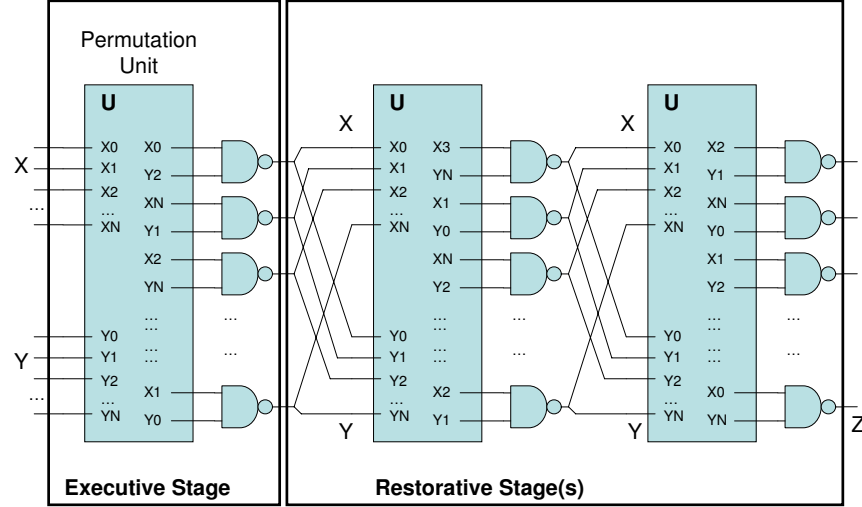


Figure 6.1: NAND Multiplexer

The *Executive Stage* contains two parts: a row of N NAND gates in parallel, and a *Permutation Unit* (i.e., block ‘U’). The initial input signals X and Y are represented by two bundles of N signals. The output of the NAND operation is the bundle Z , which will also contain N signals. Prior to the introduction of any errors, all of the signals in each bundle should match the “correct” values (i.e., $X_i = X_j \forall i, j$ and $Y_i = Y_j \forall i, j$). If errors have occurred, some fraction of these lines will contain the logical inverse of the correct value. Without loss of generality, logical true, ‘1’, is defined to be the “correct” value for X and Y , and thus ‘0’ is the correct output Z . Let (X, Y, Z) have $(kx_0 = \bar{x}N, ky_0 = \bar{y}N, kz_0 = \bar{z}N)$ stimulated signals. Thus, the triplet $(\bar{x}, \bar{y}, \bar{z})$ is the probability each variable is stimulated, while kx_0, ky_0, kz_0 represent the number of stimulated lines in each respective bundle for stage 0.

In the *permutation unit*, U , the X and Y bundles are randomly permuted and combined into N $X_i Y_j$ pairs. For example, if $N = 4$, one possible permutation is $X_2 Y_3, X_0 Y_1, X_3 Y_0, X_1 Y_2$. These XY pairs are the inputs to the N NAND gates. For the von Neumann error, each NAND gate is subject to an error which inverts the correct logical output with probability ε .

The goal of previous work [vN56, HJ02, NPK04, SNF04] was to determine the distribution of the stochastic variable \bar{z} in terms of given \bar{x} and \bar{y} . Von Neumann [vN56]

concluded that, for large N , the output probability \bar{z} is a stochastic variable with an approximately normal distribution, and the upper bound for the probability of gate failure that could be tolerated is $\varepsilon_{max} \approx 0.0107$. Recent work showed that the tolerable threshold probability for any formula constructed from NAND2 gates is $\varepsilon_{max} = (3 - \sqrt{7})/4 \approx 0.08856$ [EP98]. Beyond this (i.e., $\varepsilon > \varepsilon_{max}$), the failure probability of the NAND Multiplexer system will be larger than some fixed, positive lower bound, regardless of the bundle size N . Furthermore, for small N , the number of stimulated outputs of the executive stage is theoretically a binomial distribution, although this was disputed by [NPK04,BS04b], citing the lack of independence between the lines of the output bundle. Herein, it is proven that the lines of the output bundles are not independent, and we introduce new mathematics to accurately model the system. A combinatorics-based approach was recently used to model MAJ-3 multiplexing [RB05].

6.2.2 Han and Jonker Analytical Model. In the Han and Jonker model [HJ02], the error distribution for the NAND Multiplexing technique was developed by examining each NAND gate in the executive stage independently. The model is also used in [SNF04]. A binomial distribution described the number of asserted outputs from the executive unit, and a Markov chain modelled the output distribution after multiple stages.

The following presentation of that model follows [HJ02]. The probability of the output of a single NAND gate being stimulated is

$$\bar{z} = (1 - \bar{x}\bar{y}) + \varepsilon(2\bar{x}\bar{y} - 1). \quad (6.1)$$

This equation is valid for von Neumann errors. Other error types have a similar form. Each gate's inputs were assumed to be chosen independently. In this case, the N parallel gates function as a Bernoulli sequence. The probability distribution for

the sum of the stimulated outputs is a binomial distribution given by

$$P(k) = \binom{N}{k} \bar{z}^k (1 - \bar{z})^{N-k}, \quad k \geq 0, \quad (6.2)$$

where k is the number of stimulated outputs.

For simplicity, the input probabilities for the first stage were assumed to be equal ($\bar{x}_0 = \bar{y}_0$ and $kx_0 = ky_0 = k_0$). For each stage m , $\bar{x}_m = kx_m/N$. Thus, \bar{z}_m can be rewritten as a function of the number of stimulated X inputs and \bar{z} becomes $\bar{z}(k_m)$, the probability any NAND gate's output is stimulated (given kx_{m-1} and ky_{m-1} lines in the X and Y bundle were asserted), or

$$\bar{z}(k_{m-1}) = (1 - \varepsilon) - (1 - 2\varepsilon) \left(\frac{k_{m-1}}{N} \right)^2. \quad (6.3)$$

Next, (6.2) can be rewritten as

$$P(k_m | k_{m-1}) = \binom{N}{k_m} \bar{z}^{k_m}(k_{m-1}) (1 - \bar{z}(k_{m-1}))^{N-k_m}. \quad (6.4)$$

The probability of k_m stimulated outputs for stage m is thus

$$P(k_m) = \sum_{k_{m-1}=0}^N P(k_m | k_{m-1}) P(k_{m-1}). \quad (6.5)$$

The conditional probabilities from (6.4) can be stored in a $(N+1) \times (N+1)$ matrix Ψ as

$$\Psi = \begin{bmatrix} P(0|0) & P(1|0) & \dots & P(N|0) \\ P(0|1) & P(1|1) & \dots & P(N|1) \\ \dots & \dots & \dots & \dots \\ P(0|N) & P(1|N) & \dots & P(N|N) \end{bmatrix}. \quad (6.6)$$

Since the stages of the NAND Multiplexer are a Markov process [HJ02], the output distribution of any stage of the system is dependent only on the distribution

of the previous stage. Thus, the Ψ matrix along with the initial input distribution \mathbf{P}_0 determines the output distribution of any stage m .

Given an initial fixed input distribution

$$\mathbf{P}_0 = [p_0, p_1, p_2, \dots, p_N] \quad (6.7)$$

where p_i is the probability that i inputs are stimulated,

$$\mathbf{P}_1 = [P(0), P(1), \dots, P(N)] = \mathbf{P}_0 \Psi. \quad (6.8)$$

Generalizing for any stage m ,

$$\mathbf{P}_m = \mathbf{P}_0 \Psi^m. \quad (6.9)$$

6.2.3 Sadek, Nikolic, and Forshaw's Analytical Expression. Sadek, et al., discuss a variation on NAND Multiplexing, called Parallel Restitution [SNF04]. The authors proposed an analytical model for the number of stimulated outputs of the execution unit prior to error introduction, S_Z , and after errors are introduced, S_Q . The expression

$$P(S_Z = z) = \frac{\binom{N}{kx_m} \binom{N-z}{N-z} \binom{N-kx_m}{ky_m - N+z}}{\binom{N}{kx_m} \binom{N}{ky_m}}, \quad (6.10)$$

is directly from [vN56]. Here, kx_m and ky_m are the number of stimulated inputs in the input bundles. Without providing details, the authors consider the effects of single, double, and triple errors to obtain

$$\begin{aligned} P(S_Q = i) &= \sum_{z=0}^N P(S_Z = z) (1-\varepsilon)^{N-|z-i|} \varepsilon^{|z-i|} \\ &\quad \times \binom{\left| \frac{N}{2} \left(1 - \frac{z-i}{|z-i|} \right) - z \right|}{|z-i|}. \end{aligned} \quad (6.11)$$

These expressions are set aside in favor of the binomial model [HJ02] due to the computational complexity for larger values of N .

6.3 *Dependency Between the Outputs of the NAND Array*

This section demonstrates that the binomial distribution does not apply for $P(k_m|k_{m-1})$, since the outputs of the N NAND gates are not independent [NPK04]. The difference between “with replacement” and “without replacement” selection for X_i and Y_i is explained, and the outputs Z_i and Z_j are proven to be dependent, thus making the use of the binomial distribution invalid. These concepts lay the foundation for the development of a new model in the next section that accounts for dependence.

Consider a NAND Multiplexer composed of N parallel gates in each of M stages, where $M = 1 + 2\alpha$ and α is the number of restorative stages. Let X_i and Y_i be the events that the inputs to the i th gate of the current stage are stimulated. Likewise, Z_i is the event that the i th gate’s output is stimulated (prior to errors). Let \bar{x}_i and \bar{y}_i be the probabilities that the X_i and Y_i inputs are stimulated (i.e., logic ‘1’). The number of stimulated inputs in stage m ’s X bundle is kx_m , while the number of stimulated inputs in stage m ’s Y bundle is ky_m .

The probability that the Z_i output is stimulated is $\bar{z}_i = 1 - \bar{x}_i\bar{y}_i$. The introduction of von Neumann errors inverts each output with the probability of ε . Let E_i be the event that the i th gate in the current stage suffers a von Neumann error that inverts its output, and \bar{E}_i be the event that an error does not occur. Next, Q_i is the event that the i th gate’s output is stimulated (after errors are introduced).

Let \wedge denote set intersection. Accounting for von Neumann errors, the probability of gate \bar{z}_i ’s output being stimulated is

$$P(Q_i) = P(Z_i \wedge \bar{E}_i) + P(\bar{Z}_i \wedge E_i). \quad (6.12)$$

Using conditional probabilities, this becomes

$$P(Q_i) = P(Z_i|\bar{E}_i)P(\bar{E}_i) + P(\bar{Z}_i|E_i)P(E_i). \quad (6.13)$$

Substituting $P(Q_i) = \bar{q}_i$ and $P(\bar{E}_i) = \varepsilon$, this becomes

$$\bar{q}_i = (1 - \bar{x}_i \bar{y}_i) + \varepsilon(2\bar{x}_i \bar{y}_i - 1). \quad (6.14)$$

Assume an initial input of $kx_0 = ky_0 = 0.9N$. Thus for the first stage, $\bar{x}_i = \bar{x}_j = \bar{y}_i = \bar{y}_j, \forall i \neq j$ and (6.14) becomes

$$\bar{q} = (1 - \bar{x} \bar{y}) + \varepsilon(2\bar{x} \bar{y} - 1). \quad (6.15)$$

If $\bar{q}_i = \bar{q}_j, \forall i \neq j$ and \bar{q}_i and \bar{q}_j are independent, then S_Q , the number of stimulated outputs from this stage, has a binomial distribution with parameters N and \bar{q} .

Thus, the probability of having exactly k outputs stimulated (after errors) is

$$P(S_Q = k) = \binom{N}{k} \bar{q}^k (1 - \bar{q})^{N-k}. \quad (6.16)$$

However, \bar{q}_i and \bar{q}_j are not independent, and the binomial distribution does not apply. To prove this assertion, a preliminary theorem is presented.

Theorem 1: $P(X_i) = P(X_j), \forall i, j \in 1..N$ for both “with replacement” (WR) and “without replacement” (WOR) cases

Proof. For the WR case, the number of 1s in the N element input bundle for stage m is always kx_m . Thus $P(X_i) = kx_m/N, \forall i \in 1..N$. For the WOR case, $P(X_i)$ is

$$P(X_i) = \frac{\beta}{\chi} \quad (6.17)$$

where β is the number of arrangements of $N - 1$ elements, given that $X_i = 1$, and χ is the number of arrangement of N elements, of which kx_m are stimulated. This can be written as

$$P(X_i) = \frac{\binom{N-1}{kx_m-1}}{\binom{N}{kx_m}} = \frac{kx_m}{N}. \quad (6.18)$$

Thus, $P(X_i) = P(X_j)$ for both WR and WOR cases. \square

Theorem 2: X_i and $X_j, \forall i \neq j$, are not independent.

Proof. Proof by contradiction. If X_i and X_j were independent, then $P(X_i \wedge X_j) = P(X_i)P(X_j)$.

By Theorem 1, $P(X_i) = P(X_j) = kx_m/N$. Again using combinatorics, $P(X_i X_j) = \beta/\chi$. In this case, β is the number of arrangements of the remaining $N-2$ bits given $X_i = X_j = 1$, and χ is the number of arrangements of all N bits, of which kx_m are stimulated. This reduces to

$$P(X_i X_j) = \frac{\binom{N-2}{kx_m-2}}{\binom{N}{kx_m}} = \frac{kx_m(kx_m-1)}{N(N-1)}. \quad (6.19)$$

Clearly, $P(X_i X_j) \neq P(X_i)P(X_j)$, and thus X_i and X_j are not independent. \square

For the binomial distribution to model S_Q accurately, two conditions must be met: (1) $P(Q_i) = P(Q_j), \forall i, j$; and (2) Q_i and Q_j must be independent. Theorem 2 showed the independence requirement is not met for X_i . Next, it will be shown that independence does not hold for Z_i and ultimately Q_i .

Theorem 3: Z_i and Z_j are not independent.

Proof. As before, this is proof by contradiction. If Z_i and Z_j were independent, $P(Z_i \wedge Z_j) = P(Z_i)P(Z_j)$. First consider that

$$P(Z_i) = 1 - P(\bar{Z}_i) = 1 - P(X_i \wedge Y_i). \quad (6.20)$$

It is evident that X_i and Y_i are group-wise independent, so

$$P(Z_i) = 1 - P(X_i)P(Y_i) = 1 - \frac{kx_m}{N} \frac{ky_m}{N}. \quad (6.21)$$

Now,

$$P(Z_i \wedge Z_j) = 1 - P(\overline{Z_i \wedge Z_j}). \quad (6.22)$$

Applying De Morgan's Law, $P(\overline{Z_i \wedge Z_j}) = P(\bar{Z}_i \vee \bar{Z}_j)$,

$$\begin{aligned} P(Z_i \wedge Z_j) &= 1 - P(\bar{Z}_i \vee \bar{Z}_j) \\ &= 1 - (P(\bar{Z}_i) + P(\bar{Z}_j) - P(\bar{Z}_i \wedge \bar{Z}_j)). \end{aligned} \quad (6.23)$$

But $\bar{Z}_i \wedge \bar{Z}_j = 1$ only when $X_i = X_j = Y_i = Y_j = 1$. Thus, $P(\bar{Z}_i \wedge \bar{Z}_j) = P(X_i X_j Y_i Y_j)$. Recognizing group independence, (6.22) is separated into

$$P(\bar{Z}_i \wedge \bar{Z}_j) = P(X_i X_j) P(Y_i Y_j). \quad (6.24)$$

$P(X_i X_j)$ was derived in (6.19). Substituting kx_m with ky_m , $P(Y_i Y_j)$ takes the same form. Thus,

$$P(\bar{Z}_i \wedge \bar{Z}_j) = \frac{kx_m(kx_m - 1)}{N(N - 1)} \frac{ky_m(ky_m - 1)}{N(N - 1)}. \quad (6.25)$$

Substituting (6.25) into (6.23) results in

$$\begin{aligned}
P(Z_i \wedge Z_j) &= 1 - 2 \left(\frac{kx_m}{N} \frac{ky_m}{N} \right) \\
&\quad + \left(\frac{kx_m(kx_m-1)}{N(N-1)} \frac{ky_m(ky_m-1)}{N(N-1)} \right).
\end{aligned} \tag{6.26}$$

Now, to determine $P(Z_i)P(Z_j)$, (6.21) is used, which yields

$$\begin{aligned}
P(Z_i)P(Z_j) &= \left(1 - \frac{kx_m}{N} \frac{ky_m}{N} \right)^2 \\
&= 1 - 2 \frac{kx_m}{N} \frac{ky_m}{N} + \left(\frac{kx_m}{N} \frac{ky_m}{N} \right)^2.
\end{aligned} \tag{6.27}$$

Comparing (6.26) and (6.27), $P(Z_i \wedge Z_j) \neq P(Z_i)P(Z_j)$, and therefore Z_i and Z_j are not independent. \square

Theorem 4: Q_i and Q_j are not independent.

Proof. If Q_i and Q_j are independent, then $P(Q_i \wedge Q_j) = P(Q_i)P(Q_j)$. From (6.12),

$$\begin{aligned}
P(Q_i) &= P(Z_i \wedge \bar{E}_i) + P(\bar{Z}_i \wedge E_i) \\
&= P(Z_i)P(\bar{E}_i) + P(\bar{Z}_i)P(E_i).
\end{aligned} \tag{6.28}$$

Multiplying $P(Q_i)P(Q_j)$,

$$\begin{aligned}
P(Q_i)P(Q_j) &= [P(Z_i)P(\bar{E}_i) + P(\bar{Z}_i)P(E_i)] \\
&\quad \cdot [P(Z_j)P(\bar{E}_j) + P(\bar{Z}_j)P(E_j)] \\
&= P(Z_i)P(Z_j)P(\bar{E}_i)P(\bar{E}_j) \\
&\quad + P(\bar{Z}_i)P(Z_j)P(E_i)P(\bar{E}_j) \\
&\quad + P(Z_i)P(\bar{Z}_j)P(\bar{E}_i)P(E_j) \\
&\quad + P(\bar{Z}_i)P(\bar{Z}_j)P(E_i)P(E_j).
\end{aligned} \tag{6.29}$$

Derivation of $P(Q_i Q_j)$ proceeds similarly as

$$\begin{aligned}
P(Q_i Q_j) &= P(Z_i Z_j \bar{E}_i \bar{E}_j) + P(\bar{Z}_i Z_j E_i \bar{E}_j) \\
&\quad + P(Z_i \bar{Z}_j \bar{E}_i E_j) + P(\bar{Z}_i \bar{Z}_j E_i E_j) \\
&= P(Z_i Z_j) P(\bar{E}_i) P(\bar{E}_j) \\
&\quad + P(\bar{Z}_i Z_j) P(E_i) P(\bar{E}_j) \\
&\quad + P(Z_i \bar{Z}_j) P(\bar{E}_i) P(E_j) \\
&\quad + P(\bar{Z}_i \bar{Z}_j) P(E_i) P(E_j). \tag{6.30}
\end{aligned}$$

From Theorem 3, $P(Z_i Z_j) \neq P(Z_i)P(Z_j)$, and thus in general (6.29) \neq (6.30). It is not necessary to prove (6.29) \neq (6.30) for all combinations of kx , N , and ε . As a counterexample, using typical values of $N = 100$, $\varepsilon = 0.01$, and $kx_0 = ky_0 = 0.9N$, $P(Q_1 Q_2) \approx 0.037$ and $P(Q_1)P(Q_2) \approx 0.038$. Thus, $P(Q_1 Q_2) \neq P(Q_1)P(Q_2)$ and Q_i and Q_j are not independent. \square

6.4 Updated Distribution Model

Since Q_i and Q_j are not independent, the binomial distribution should not be used to model $P(S_Q = i)$. In this section, a new model is derived that accounts for the dependency.

6.4.1 Derivation of S_Z . To determine $P(S_Q = i)$, the number of stimulated outputs after errors are introduced, $P(S_Z = i)$ is first derived, followed by a summation of conditional probability expressions. It is easier to work with the number of zeros in the output rather than the number of ones (i.e., using NAND gates, there is only one combination of X and Y inputs that yields $Z = 0$, whereas there are three combinations of inputs that yield $Z = 1$. By counting with the number of zeros instead of ones, there will be fewer terms in the summation expression). Thus, S_C is

the number of non-stimulated outputs, or

$$S_C = \sum_{i=1}^N \bar{Z}_i. \quad (6.31)$$

Note that $S_C = N - S_Z$ and $P(S_C = i) = P(S_Z = N - i)$. This will be useful later. $P(S_C = i)$ is the number of arrangements of the X and Y bundles resulting in $S_C = i$ divided by the total number of arrangements of the X and Y bundles.

For the denominator, each bundle has N inputs, of which kx_m and ky_m are stimulated. The number of arrangements of the X bundle, β_x , is

$$\chi_x = \binom{N}{kx_m}. \quad (6.32)$$

The number of arrangements of the Y bundle, χ_y , is derived similarly.

Since X and Y are group-wise independent, the total number of arrangements of both bundles is simply the product of χ_x and χ_y , or

$$\chi = \binom{N}{kx_m} \binom{N}{ky_m}. \quad (6.33)$$

To compute the numerator, β , observe that for $Z_i = 0$ a NAND gate requires that $X_i = Y_i = 1$. Computing the number of ways to arrange the kx_m stimulated inputs in the X bundle is done in the same way as in (6.32). Having fixed the X arrangements, look inside the group of kx_m logic ones. For $P(S_C = i)$, i of the Y inputs corresponding to this group must be logic one. Thus the number of arrangements of Y is

$$\beta_{Y_{inX1}} = \binom{kx_m}{i}. \quad (6.34)$$

Now consider the remaining $N - kx_m$ members of the Y bundle, of which $ky_m - i$ of the inputs are logic one. The number of arrangements of this subgroup is

$$\beta_{YinX0} = \binom{N - kx_m}{ky_m - i}. \quad (6.35)$$

The entire numerator, β , is thus the product of (6.32), (6.34), and (6.35). The denominator is (6.33) and the complete equation for $P(S_C = i)$, the number of non-stimulated (i.e., logic zero) outputs, is

$$P(S_C = i) = \frac{\binom{N}{kx_m} \binom{kx_m}{i} \binom{N - kx_m}{ky_m - i}}{\binom{N}{kx_m} \binom{N}{ky_m}}. \quad (6.36)$$

Observing that $P(S_C = i) = P(S_Z = N - i)$, the equation for $P(S_Z = i)$ becomes

$$P(S_Z = i) = \frac{\binom{N}{kx_m} \binom{kx_m}{N - i} \binom{N - kx_m}{ky_m - N + i}}{\binom{N}{kx_m} \binom{N}{ky_m}} \quad (6.37)$$

for $N - i \leq kx_m$ and $ky_m - N + i \leq N - kx_m$. Otherwise, $P(S_Z = i) = 0$.

The derivation is similar to other models. $P(S_C)$ corresponds to ρ in [vN56]. However, von Neumann goes on to approximate ρ for large N and the derivations diverge at this point. $P(S_Z)$ corresponds to $P(w|e_1, e_2)$ for MAJ-3 Multiplexing in [RB05].

6.4.2 Derivation of the distribution of S_Q . Having derived an expression for the output distribution prior to the injection of the von Neumann errors, we now build an expression for the distribution of S_Q , the number of stimulated outputs after errors are injected, by summing conditional probabilities.

To compute $P(S_Q = i)$, it is necessary to account for all of the combinations of X and Y bundles, as well as von Neumann errors, that result in $S_Q = i$. For example, $S_Q = i$ can be obtained when $S_Z = i$ and no errors change the output bits. It is also possible to obtain $S_Q = i$ when $S_Z = i - 1$ and there is one von Neumann error on

a $Z_k = 0$ output that results in $Q_k = 1$ (thereby increasing the count S_Q by one). It is also possible to obtain $S_Q = i$ when $S_Z = i + 1$ and one von Neumann error on a $Z_k = 1$ output results in $Q_k = 0$ (decreasing the count S_Q by one). Furthermore, the same result can be obtained with larger numbers of errors. Let S_P be the number of outputs $Z_k = 0$ suffering from a von Neumann error yielding $Q_k = 1$. Similarly, S_N is the number of outputs $Z_k = 1$ suffering from a von Neumann error yielding $Q_k = 0$. In general,

$$S_Q = S_Z + S_P - S_N. \quad (6.38)$$

$P(S_Q = i)$ can be written as

$$\begin{aligned} P(S_Q = i) = & P(S_Z = 0 \wedge S_N = 0 \wedge S_P = i) \\ & + P(S_Z = 0 \wedge S_N = 1 \wedge S_P = i + 1) \\ & + \dots \\ & + P(S_Z = 0 \wedge S_N = N - i \wedge S_P = N) \\ & + P(S_Z = 1 \wedge S_N = 0 \wedge S_P = i - 1) \\ & + P(S_Z = 1 \wedge S_N = 1 \wedge S_P = i) \\ & + \dots \\ & + P(S_Z = 1 \wedge S_N = N - i + 1 \wedge S_P = N) \\ & + \dots \\ & + P(S_Z = N \wedge S_N = N - i \wedge S_P = 0). \end{aligned} \quad (6.39)$$

Many of the terms in (6.39) are zero. For example, $P(S_Z = k \wedge S_N = l \wedge S_P = m) = 0$ when $S_N > S_Z$ or $S_P > N - S_Z$. $P(S_Z = k \wedge S_N = l \wedge S_P = m)$ can be specified using conditional probabilities as

$$P(S_Z = k \wedge S_N = l \wedge S_P = m) = \quad (6.40)$$

$$P(S_Z = k) P(S_N = l \wedge S_P = m | S_Z = k). \quad (6.41)$$

Suppose that errors occur on the Z_i outputs independently. The probability of von Neumann error is fixed for all Z_i as ε and occurs independently on $Z_i = 1$ bits to create negative errors (i.e. logic one bits flipped to logic zero), and on $Z_i = 0$ bits to create positive errors (i.e., logic zeros flipped to become logic one). Since ε is fixed for each gate, the binomial distribution applies. Thus, (6.41) can be written as

$$\begin{aligned}
P(S_Z=k \wedge S_N=l \wedge S_P=m) &= \\
P(S_Z=k) P(S_N=l | S_Z=k) P(S_P=m | S_Z=k) &= \\
P(S_Z=k) \binom{k}{l} \varepsilon^l (1-\varepsilon)^{k-l} \binom{N-k}{m} \varepsilon^m (1-\varepsilon)^{N-k-m}. &
\end{aligned} \tag{6.42}$$

Finally, (6.39) can be written as a summation where

$$\begin{aligned}
P(S_Q=i) &= \\
\sum_{j=0}^N \sum_{k=0}^j P(S_Z=j) P(S_N=k \wedge S_P=i+k-j | S_Z=j). &
\end{aligned} \tag{6.43}$$

Note that $P(S_Q=i) = 0$ when $S_N > kz_m$, $S_P > N - kz_m$, or $i + k - j < 0$.

A simple transformation of (6.43) yields $P(S_{Q_m}|S_{Q_{m-1}})$, which is used to compute the elements in the Ψ matrix. The equation is

$$\begin{aligned}
P(kq_m=i | kq_{m-1}) &= P(S_{Q_m}=i | S_{Q_{m-1}}) = \\
\sum_{j=0}^N \sum_{k=0}^j P(S_Z=j | S_{Q_{m-1}}) P(S_N=k \wedge S_P=i+k-j | S_Z=j) &
\end{aligned} \tag{6.44}$$

Table 6.1: NAND Truth Table for Relevant Errors

Inputs		Outputs					
X	Y	Z	VN	oSA0	oSA1	iSA0	iSA1
0	0	1	0	0	1	1	1
0	1	1	0	0	1	1	0
1	0	1	0	0	1	1	1
1	1	0	1	0	1	1	0

where $P(S_{Z_m}|S_{Q_{m-1}})$ is a generalization of (6.37). Observing that $kx_m = kq_{m-1} = S_{Q_{m-1}}$, $P(S_{Z_m}|S_{Q_{m-1}})$ can be written as

$$P(S_{Z_m} = i|S_{Q_{m-1}}) = \frac{\binom{N}{kx_m} \binom{kx_m}{N-i} \binom{N-kx_m}{ky_m-N+i}}{\binom{N}{kx_m} \binom{N}{ky_m}}. \quad (6.45)$$

Note that $P(S_{Q_m}|S_{Q_{m-1}})$ corresponds to $P(s|w)$ in [RB05]. The authors' definition for error is one that inverts the correct output, just as in this chapter. However, the equation for $P(s|w)$ does not account for positive errors (i.e., logic zeros flipped to logic ones, or S_P). The equation for $P(s|w)$ is easily fixed using the method described in this section.

6.5 New Fault Types: Input and Output Stuck-At Faults

Having derived an expression for $P(S_Q = i)$ for von Neumann faults, expressions for other types of logical errors can be formulated. Output stuck-at errors, when they occur, force the output of the NAND gate to either logic one (oSA1) or zero (oSA0). Input stuck-at errors can occur on either the X or the Y input. This section considers the case of a single input error on the gate's X input. Table 6.1 shows the truth table for the error-free NAND gate (column Z), as well as erroneous outputs obtained under the five error types.

6.5.1 Output Stuck-At Faults. The method of deriving $P(S_Q = i)$ for the two output stuck-at faults is similar to that used for the von Neumann error. Since

$S_P = 0$ for oSA0 errors, (6.38) can be simplified. Likewise, for oSA1 errors, $S_N = 0$ and many of the terms in (6.39) are zero. For the oSA0 error, (6.39) becomes

$$\begin{aligned}
P(S_Q=i) &= P(S_Z=i \wedge S_N=0) \\
&+ P(S_Z=i+1 \wedge S_N=1) \\
&+ \dots \\
&+ P(S_Z=N \wedge S_N=N-i). \tag{6.46}
\end{aligned}$$

For the oSA1 error, (6.39) becomes

$$\begin{aligned}
P(S_Q=i) &= P(S_Z=i \wedge S_P=0) \\
&+ P(S_Z=i-1 \wedge S_P=1) \\
&+ \dots \\
&+ P(S_Z=0 \wedge S_N=i). \tag{6.47}
\end{aligned}$$

As in the case of the von Neumann error, the errors are assumed to be independent of the X and Y inputs, and thus (6.46) and (6.47) can be separated into conditional probabilities as

$$\begin{aligned}
P(S_Z=k \wedge S_N=l) &= \\
P(S_Z=k) P(S_N=l | S_Z=k) &= \\
P(S_Z=k) \binom{k}{l} \varepsilon^l (1-\varepsilon)^{k-l} & \tag{6.48}
\end{aligned}$$

for oSA0, and

$$\begin{aligned}
P(S_Z=k \wedge S_P=m) &= \\
P(S_Z=k) P(S_P=m | S_Z=k) &= \\
P(S_Z=k) \binom{N-k}{m} \varepsilon^m (1-\varepsilon)^{N-k-m} & \tag{6.49}
\end{aligned}$$

for oSA1.

Finally, (6.46) and (6.47) can be written as a summation as

$$\begin{aligned}
P(S_Q = i) &= \\
\sum_{j=0}^N P(S_Z = j) P(S_N = j - i | S_Z = j) &= \\
\sum_{j=i}^N P(S_Z = j) P(S_N = j - i | S_Z = j) & \quad (6.50)
\end{aligned}$$

for oSA0, and

$$\begin{aligned}
P(S_Q = i) &= \\
\sum_{j=0}^N P(S_Z = j) P(S_P = i - j | S_Z = j) &= \\
\sum_{j=0}^i P(S_Z = j) P(S_P = i - j | S_Z = j) & \quad (6.51)
\end{aligned}$$

for oSA1.

The conditional probabilities $P(S_Q = i | S_{Q_{m-1}})$ used in Ψ are derived from (6.50) and (6.51) by replacing $P(S_Z = j)$ with $P(S_Z = j | S_{Q_{m-1}})$ as was done for (6.45).

6.5.2 Input Stuck-At-Zero. As shown in Table 6.1, the Q output of the iSA0 error matches that of the oSA1 error. The faults affect the circuit in the same way. Thus, $P(S_Q = i)$ for the iSA0 fault is also (6.51).

6.5.3 Input Stuck-At-One. Computation of $P(S_Q = i)$ for the iSA1 error is much more complicated than for the other cases. The technique described for the previous errors computed $P(S_Z = i)$ using a combinatorics method (6.37), and then applied the effects of error. For the iSA1 error, the combinatorics approach cannot be used, since errors occur prior to the permutation unit, and thus the number of X and Y inputs arriving at the NAND gates are not known.

The probability that any particular output will be stimulated is

$$P(Q_i) = P(Z_i \wedge \bar{E}) + P(\bar{Y}_i \wedge E). \quad (6.52)$$

Although this value is constant for all i , $P(Q_i) = P(Q_j), \forall i \neq j$, the inputs are not independent, and therefore the resulting distribution for S_Q is not a binomial distribution.

While a solution will not be presented for this problem, some work has been done in the area of dependent Bernoulli trials that may be applicable. One avenue of investigation uses the multivariate characteristic function [Spr79] to develop a probability density function. Another paper proposes a full-likelihood procedure [GB95]. Both of these techniques generate a large number of conditional probability terms as the bundle size N increases, and are not computationally feasible. For the present, simulation techniques produce acceptably accurate results in a reasonable time.

6.6 *Simulation*

To test the validity of the improved model for von Neumann faults, as well as the new models for oSA0, oSA1, and iSA0 faults, a MATLAB simulation was constructed. This section describes the simulation, and compares the accuracy of the new models against the MATLAB simulation, as well as the two analytical models [HJ02, SNF04] and the probabilistic model checker PRISM [NPK04, PNK04, BS04b].

6.6.1 Simulation Setup. The MATLAB simulation duplicates the operation of a NAND Multiplexer for a large number of iterations to compute desired test statistics. For each iteration, an initial N element vector is created for the X and the Y inputs. Assuming $\bar{x} = 0.9$, $kx_0 = 0.9N$ of the elements in each bundle are stimulated (i.e., ‘1’). The initial vectors \vec{X}_0, \vec{Y}_0 are randomly permuted. For the von Neumann error, and the two output stuck-at errors, the NAND operation is performed on each pair of elements in the two vectors to create an output vector \vec{Z}_0 . Next, an N

element error vector is created, with the probability ε that each element is in error. The error vector is combined with the ‘correct’ output vector \vec{Z}_0 to create the final output vector \vec{Q}_0 , either by inverting the elements of \vec{Z}_0 that suffer errors (in the case of von Neumann errors), or forcing their values to either ‘1’ or ‘0’ in the case of the output stuck-at faults. For the input stuck-at-0 error, the error vector is applied to the \vec{Y}_0 vector prior to the NAND operation.

The sum of the stimulated outputs in \vec{Q}_0 , $kz_0 = i$, is computed and the appropriate bin in a histogram variable ($total_i, i \in 0..N$) is incremented. When the simulation is complete, the histogram array variable is divided (element by element) by the number of iterations to yield the final probability density function for that particular stage.

The operation of the MATLAB simulation’s permutation unit, U, for the ‘with replacement’ case used by the original analytical model [HJ02], and the ‘without replacement’ case used by the improved analytical model and the PRISM simulation [NPK04], is different. For the WR case, \vec{X}_m and \vec{Y}_m vector inputs to later stages, m , are determined element by element, with $\bar{x}_{m,j} = kx_m/N, j \in 1..N$ being the probability that the j th input line in the \vec{X} vector for stage m is stimulated, given that kz_{m-1} outputs from the previous stage were stimulated. For example, given $N = 20$, if the first NAND stage results in $kz_0 = 16$, then $\bar{x}_{1,j} = 16/20, \forall j \in 1..N$. Since each bit is determined randomly, kx_1 may not equal kz_0 . Instead, kx_1 has a binomial distribution with a mean of kz_0 .

For the WOR case, the \vec{Z}_{m-1} vector from the previous stage, $m-1$, is permuted to create the new vectors \vec{X}_m and \vec{Y}_m . Thus $kz_{m-1} = kx_m = ky_m$. This important distinction is the cause of the difference between the results of the WR and WOR cases.

In addition to the output probability density function for each stage, the MATLAB simulation computes the expected percentage of incorrect outputs (EPIO), and the probability that the number of errors is less than 10% of the bundle (PEL10).

The EPIO is simply the mean of \bar{z} for stage m . PEL10 is the sum of the probabilities

$$PEL10_m = P(k_m < 0.1N) = \sum_{j=0}^{\lfloor 0.1N \rfloor} P_m(j). \quad (6.53)$$

Simulations were run over a range of error probabilities ε , as well as bundle sizes N (e.g., 20, 40, 100), number of stages M (3 to 15), and error types (i.e., ‘von Neumann’, ‘iSA0’, ‘oSA1’, and ‘oSA0’). For each case, the analytical predictions are calculated using the Han and Jonker binomial technique [HJ02], Sadek’s analytical expression [SNF04], and the combinatorial technique proposed herein.

For most combinations of parameters, the simulation ran for 20,000 iterations. Each run of 20,000 iterations was broken into 100 groups of 200 iterations to construct 95% confidence intervals on the estimates for the PDF and PEL10 statistics. The choice of 20,000 iterations was sufficiently large to yield acceptably small confidence interval for each statistic in most cases. For several cases, trials of 2 million iterations were run to obtain tighter confidence intervals. To compute the confidence intervals for the PDF, the sample mean, \bar{p} , and variance, s^2 , for every $P(S_Q = i)$ was computed for each of the 100 groups. Since the number of groups is large, it is assumed the mean values follow a normal distribution with unknown mean and variance. The $100(1 - \alpha)\%$ confidence interval, $\bar{p} \pm E$, for the mean of $(P(S_Q = i))$ is computed using the Student’s t distribution [All90] where

$$E = t_{N-1, \alpha/2} \times \frac{s}{\sqrt{N}}. \quad (6.54)$$

The PRISM probabilistic model checker [PNK04, BS04a] was also used to compute PDF and PEL10 estimates. A PRISM model consists of a description of the system to be modelled, as well as a set of properties to be checked against the model. The model and its operation are described in [NPK04]. The default run-time parameters were used for the PRISM program. Where practical, the same scenarios were

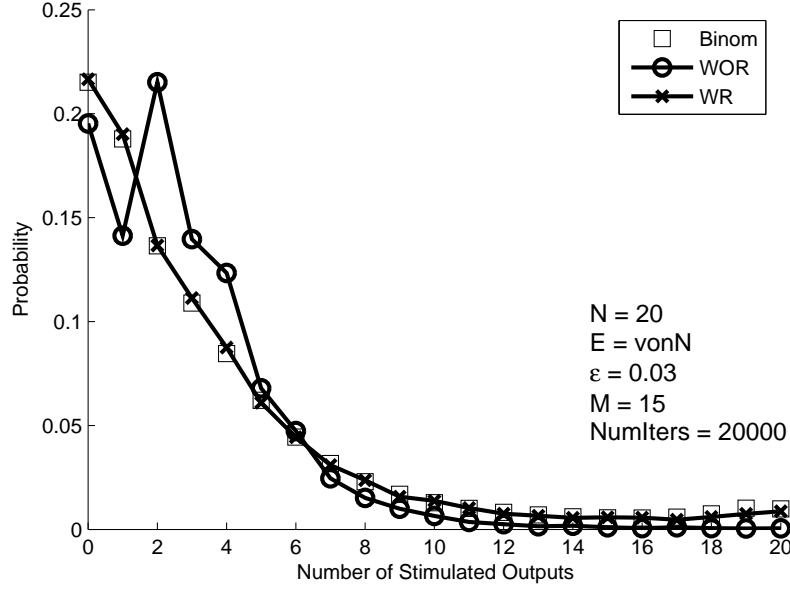


Figure 6.2: Output PDF for 7th Restorative Stage($M=15$) for $N=20$ [WR vs. WOR models]

used in both the PRISM simulation and the MATLAB simulation, and their results compared. The results of the simulations are discussed in the next section.

6.6.2 Simulation Results.

6.6.2.1 von Neumann Case. The output distribution for the seventh restorative stage (i.e., $M=15$) for $N = 20$ and $\varepsilon = 0.03$ is shown in Figure 6.2. Clearly, there is a difference between the WR and WOR results. The analytical prediction from [HJ02] (labelled ‘Binom’) closely matches the WR case. Figure 6.3 shows the results for the new analytical predication and the PRISM simulation for the same scenario. The combinatoric model proposed in this chapter (labelled ‘Combin’) closely matches the WOR case. PRISM approximates the WOR case, but some differences can be observed.

The probability that the number of errors will be less than 10% (PEL10) of the bundle size for one restorative stage (i.e., $M=3$), $N = 40$, and von Neumann errors, is shown in Figure 6.4. The PEL10 data versus the number of stages (M) is

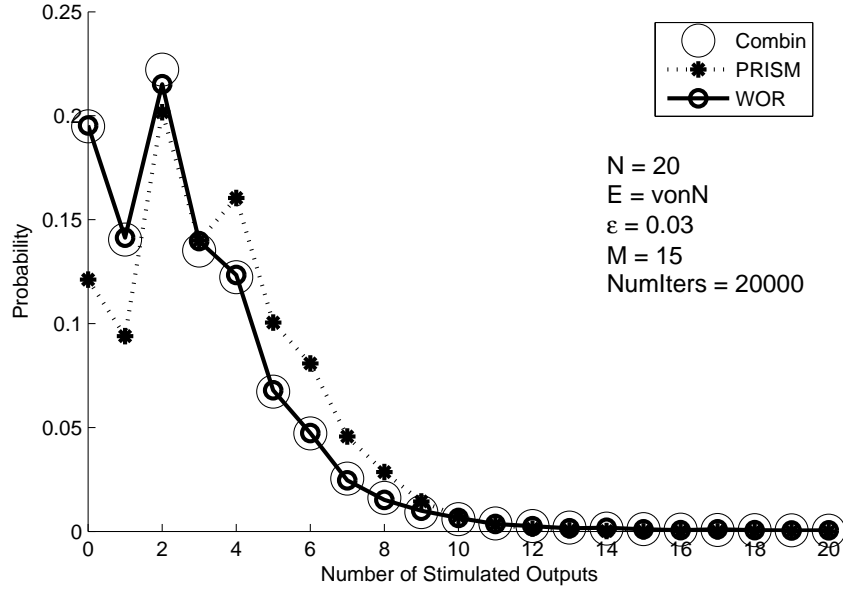


Figure 6.3: Output PDF for 7th Restorative Stage(M=15) for N=20 [PRISM and Combin. Models]

shown in Figure 6.5. The binomial model from [HJ02] follows the WR simulation, while the combinatoric model correctly follows the WOR curve. Figure 6.6 shows the 95% confidence intervals for reliability. Both analytical predictions fall within the 95% confidence intervals of the modelled system (i.e., WR or WOR). The PEL10 parameter is important to modelling the overall reliability of a processor using NAND Multiplexing. It will be shown later that even small errors in this value have a huge impact in estimating the level of redundancy necessary to achieve a specified reliability.

The next three plots (Figures 6.7, 6.8, and 6.9) show the differences between the WOR simulation, Sadek's analytical expressions, the PRISM simulation, and the new analytical model. Longer runs of 2 million iterations are used, and the resulting 95% confidence intervals are overlaid. For all three statistics, the combinatoric model produces values within the 95% confidence interval of the simulation. Both Sadek's analytical model and the PRISM simulation produced results outside the confidence interval.

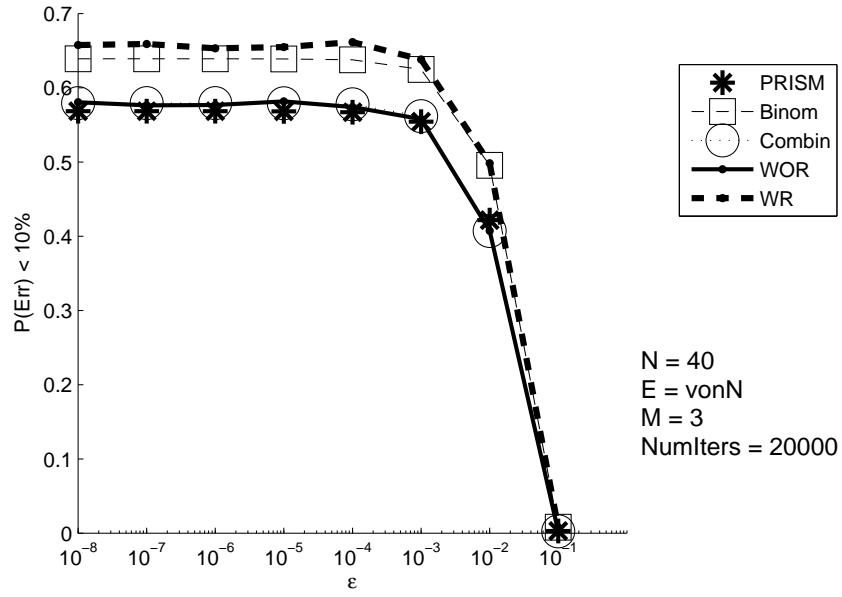


Figure 6.4: Reliability vs Error Probability ($N=40$)

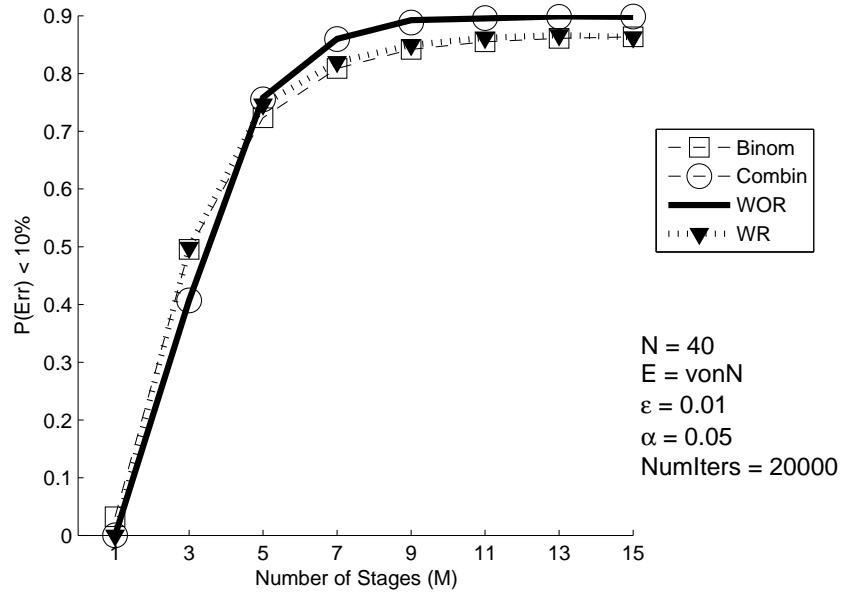


Figure 6.5: Reliability vs Number of Stages ($N=40$)

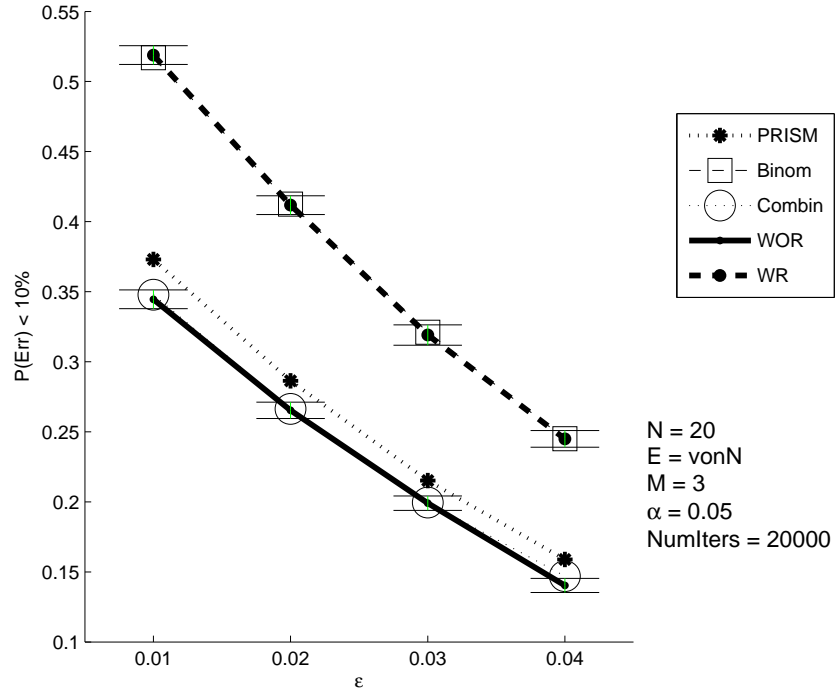


Figure 6.6: Reliability vs Error Probability ($N=20$)

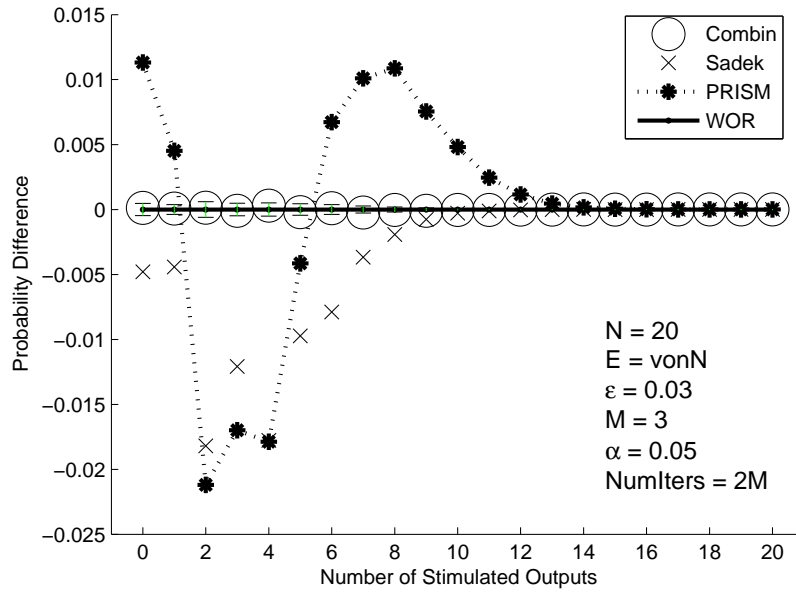


Figure 6.7: Output PDF Residuals 1st Rest. Stage ($N=20$)

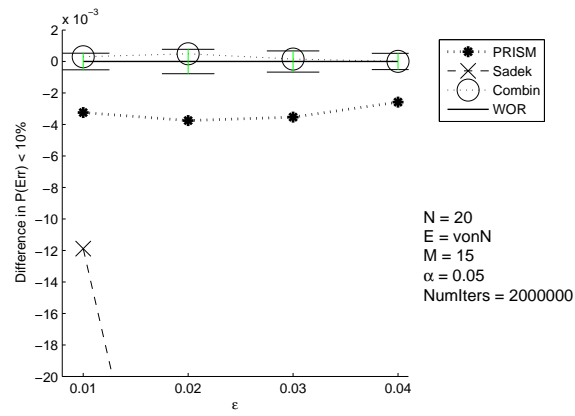


Figure 6.8: Residual Reliability vs. Error Probability ($N=20$)

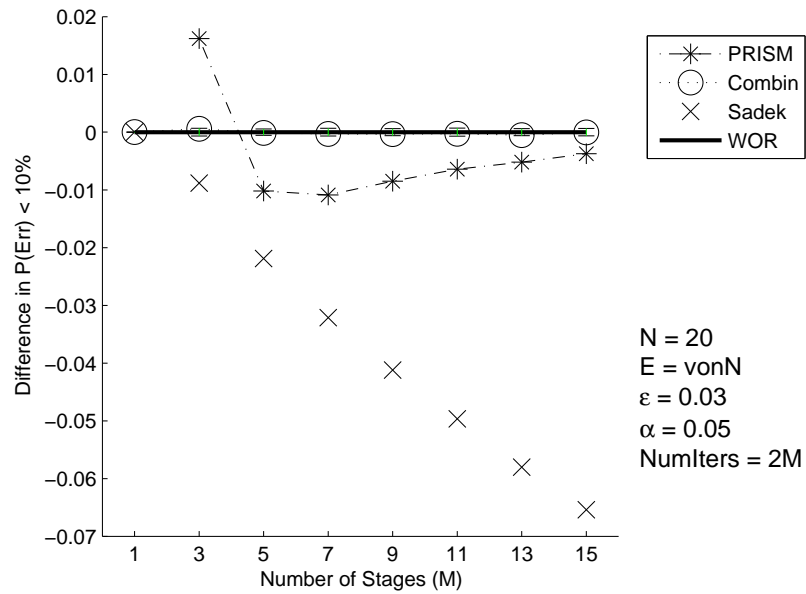


Figure 6.9: Residual Reliability vs. Number of Stages ($N=20$)

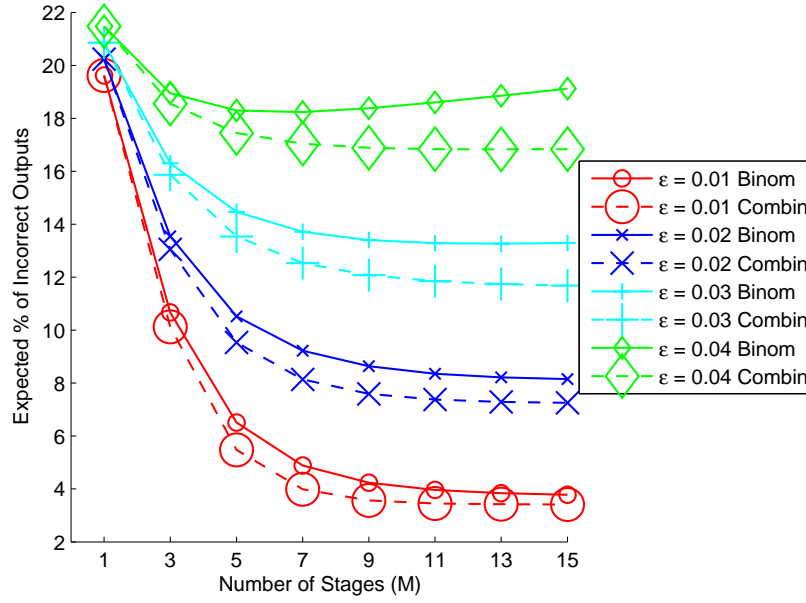


Figure 6.10: Expected Percentage of Incorrect Outputs (N=20)

As originally discussed in [NPK04], the expected percentage of incorrect outputs shows a divergence between the WR and WOR models. Figure 6.10 shows the original [HJ02] and improved analytical models. Although not shown, the analytical models closely match the simulation results for the WR and WOR cases respectively. The curves in this figure show that the original model can often underestimate the reliability of the NAND Multiplexer, particularly as the number of stages and the error probability increases.

In summary, for the von Neumann error it has been shown that incorporating the dependence between the outputs of the NAND gates produces a more accurate model. Further, the simulation validates the Markov relationship proposed in Han and Jonker's original model. Figures 6.11 and 6.12 show the reliability of the NAND Multiplexer based on the improved model.

6.6.2.2 Stuck-At Faults. The simulation results for the other error types validated the improved analytical model. The analytical model from [HJ02]

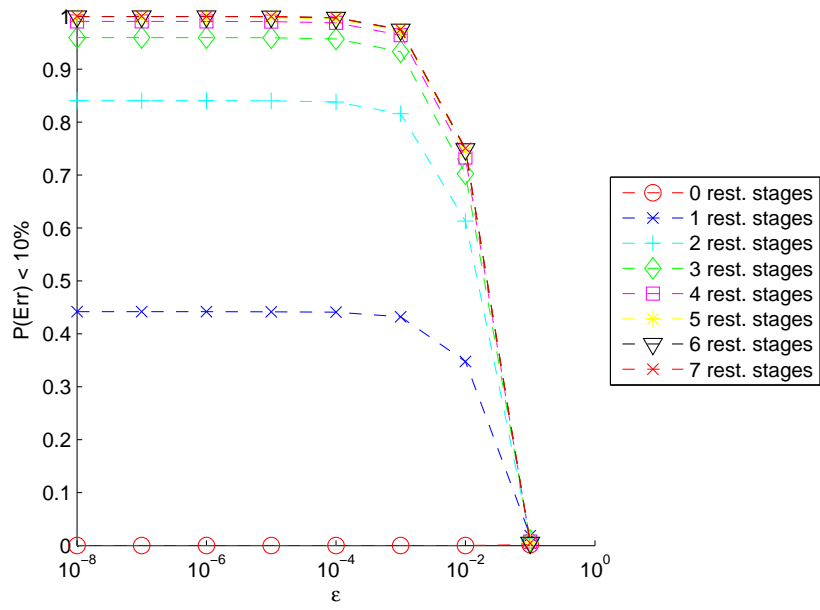


Figure 6.11: Reliability vs Error Probability (N=20)

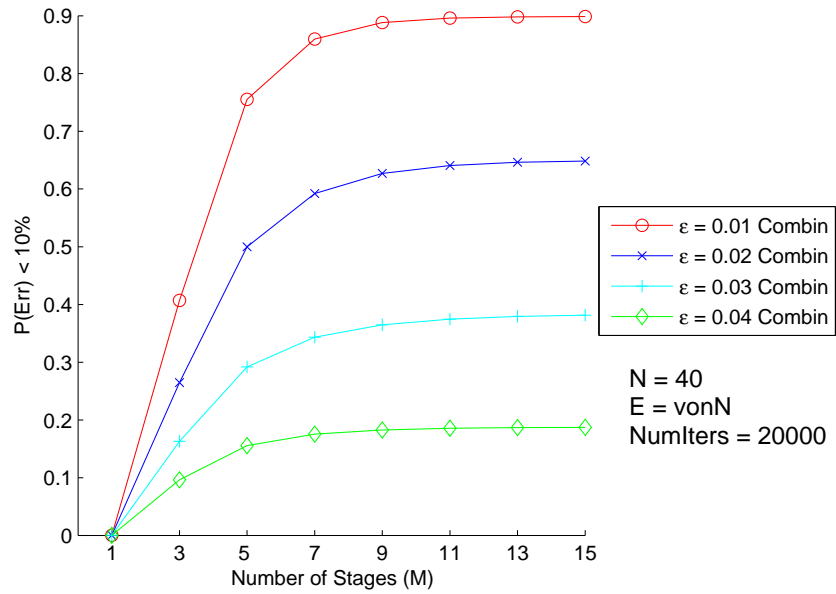


Figure 6.12: Reliability vs Number of Stages (N=20)

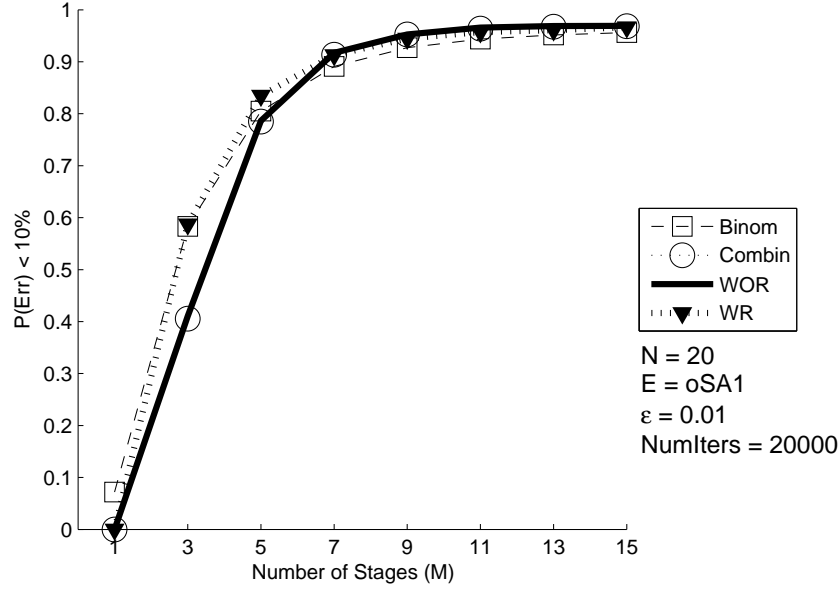


Figure 6.13: Reliability vs Number of Stages (oSA1)

was used after substituting the new equations for \bar{z} for the three error types (oSA0, oSA1, and iSA0). As before, the original analytical model was found to match the WR case, while the combinatoric model correctly follows the WOR case.

The results for the Output Stuck-At-One fault are shown in Figure 6.13. The plot of PEL10 vs. the number of stages, M , for oSA1 is similar to Figure 6.12 and thus omitted. The results for the Output Stuck-At-Zero fault are shown in Figures 6.14 and 6.15. The results for the Input Stuck-At-Zero fault were identical to those of the oSA1 error.

6.6.2.3 Algorithmic Complexity. Approximate run times for the various methods are shown in Table 6.2. The simulations were run on a 2.8GHz Pentium 4 with Hyperthreading enabled and 2GB of RAM. While not shown in Table 6.2, the Ψ matrix for $N = 100$ can be computed in roughly 3.5 hours. Computing the PDF for such a large bundle size in PRISM would require much longer. Therefore, the combinatorics-based analytical model becomes significantly faster relative to PRISM as bundle size N increases.

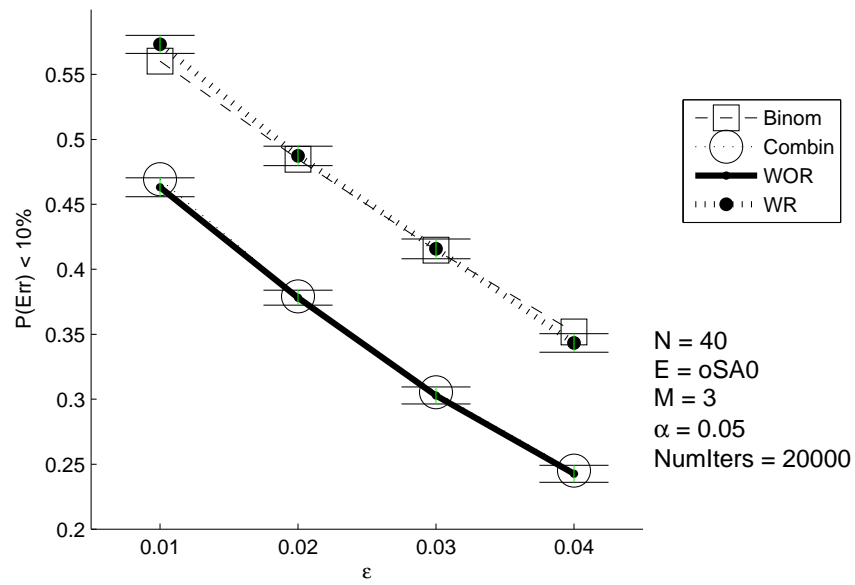


Figure 6.14: Reliability vs Error Probability (oSA0)

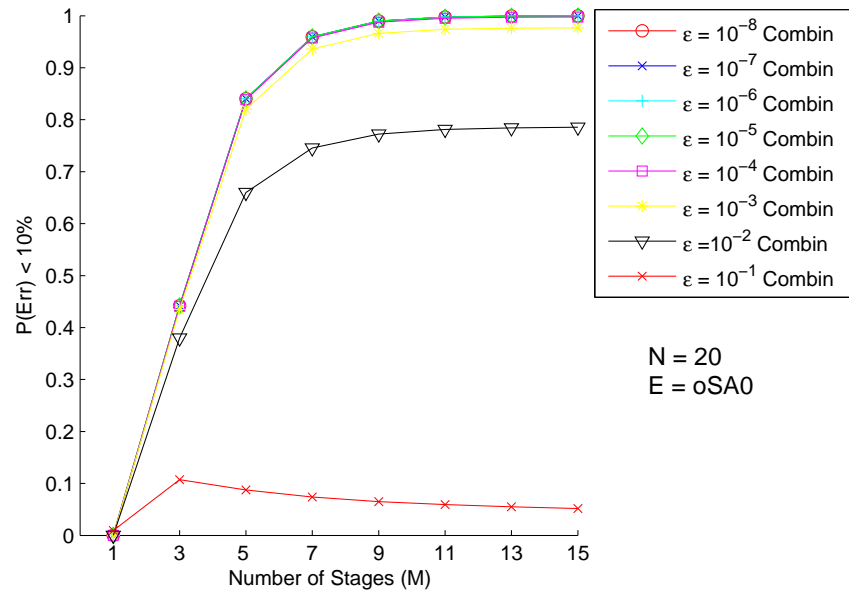


Figure 6.15: Reliability vs Number of Stages (oSA0)

Table 6.2: PDF Computation Times for $N = 40, M = 15$

Simulation	vN	oSA0	oSA1
JH	< 1 min	< 1 min	< 1 min
GR	5 min	2 min	2 min
PRISM	8 hrs		
MATLAB Sim (20K iterations)	20 min	20 min	20 min

6.7 An Application

NAND Multiplexing can be used to design fault tolerant nanoelectronic systems. When nanoelectronic devices such as single electron transistors or carbon nanotube transistors become feasible to mass produce, it will be possible to construct microchips with large numbers of devices ($10^{10} - 10^{12}$), albeit with higher device failure rates than in silicon CMOS. These additional devices can be used for redundancy-based fault tolerance techniques such as NAND Multiplexing. The overall system might use fault tolerance techniques such as NAND Multiplexing as well as reconfiguration to overcome transient and permanent errors that occur during operation. To support reconfiguration, a fault tolerant processor would likely be constructed from an array of simple processing elements. The processor would be mapped onto a portion of the nodes, and reconfigured if any node suffers permanent failure. Both reconfiguration and NAND Multiplexing require large numbers of redundant devices, and the accurate estimation of the required redundancy necessary to achieve a desired reliability will be crucial in the design of these systems.

Accurate computation of the reliability of NAND Multiplexing is key, as small errors can have a huge impact on the estimate for overall system reliability when massive numbers of devices are used. Consider a simple ($l = 32$)-bit processor composed of $n = 1000$ processing nodes [HJ02]. The underlying device technology results in errors with probability $\varepsilon = 10^{-4}$. The desired overall reliability of the system is $R_c \geq 0.9$. Each processor node contains $M = 15$ stages of logic and incorporates NAND Multiplexing. The designer wants to determine whether a bundle size of $N = 50$ is sufficient to guarantee the required reliability. Reliability of a single bit after $M = 15$ stages is

PEL10. Reliability of a l -bit processing node is thus

$$R_p = PEL10^l. \quad (6.55)$$

Since the chip has n nodes, overall reliability is

$$R_c = R_p^n. \quad (6.56)$$

The *PEL10* value calculated using the binomial method is $PEL10_{JH} = 0.999254763$, and using the combinatorics method is $PEL10_{GR} = 0.9999999847$. Although the values are only 0.08% different, when raised to the nl -th power, the difference in the resulting reliability estimates is significant. For the binomial method, $R_c \approx 4.3577 \times 10^{-11}$, while for the combinatoric method $R_c = 0.99511$. In this case, use of the binomial method leads to the incorrect conclusion that the system does not meet the reliability requirement. The designer may unnecessarily increase the bundle size, adding to the hardware cost of the design. Calculating *PEL10* for increasing bundle sizes until $R_c > 0.9$, the binomial prediction requires $N \geq 88$, while the combinatoric method requires only $N \geq 45$. Thus, the combinatoric method achieves the required reliability with approximately half the hardware.

6.8 Conclusion

A new method is proposed to predict the reliability of NAND Multiplexing, a fault-tolerance technique based on large-scale duplication of fault-prone devices. A combinatorics-based method was used to derive the output distribution for moderate bundle sizes, N . This technique represents the first accurate analytic model for NAND multiplexing for moderate amounts of redundancy, improving on previous methods by accounting for dependence between the signals of the input bundles.

Three new errors are modelled: Output Stuck-At Zero, Output Stuck-At One, and Input Stuck-At One. A MATLAB simulation validated the combinatorics model

for the NAND Multiplexer and confirmed the correctness of the new model for all four error types. The Markov nature of the underlying system proposed in [HJ02] was validated. The analytical model produces PEL10 results that are up to 20% more accurate than those from probabilistic model checkers [NPK04, BS04a], in less than one tenth the time.

NAND Multiplexing has a variety of uses in nanotechnology, fault and defect tolerant computing, and in space applications. NAND Multiplexing is one technique to achieve reliable computation with unreliable component devices. The model proposed in this chapter can be used to make intelligent design decisions between redundancy and reliability to achieve acceptable reliability with the smallest number of components.

While NAND Multiplexing has not been widely used due to its large overhead requirements, this chapter has shown that it can function reliably at high defect rates (i.e., $\lambda_1 > 10^{-5}$). Thus, NAND Multiplexing may see use in future nanodevice technologies in which small size makes the overhead requirement acceptable. The work of this chapter makes it possible to accurately model the performance of NAND Multiplexing in the range of highest interest.

VII. High Level Architectures

7.1 *Introduction*

This chapter examines the high level architecture of a fault and defect tolerant CPU. It lists the desired, and often competing, performance characteristics of the CPU and how the various fault tolerance techniques discussed in Chapter II can be applied. A concept of operations for the fault and defect tolerant computer is proposed, outlining manufacturing testing, startup configuration, runtime testing and recovery, and soft error detection and recovery. This chapter provides the general framework from which the cache and core CPU architectures will be developed in the next chapters.

7.2 *Desired Characteristics*

The fault and defect tolerant CPU should have the following characteristics:

- High manufacturing yield.
- High long term reliability.
- Resistant to soft errors and single event upsets
- Testable.
- Low Power.
- Minimal Redundancy.
- High Speed.

The FDT computer must be able to run operating system and application software reliably when constructed from devices that have manufacturing defects, operational failures, and soft errors at a rate much higher than a “conventional” computer. It should have an acceptable manufacturing yield. Conventional computers achieve this through careful manufacturing process control, which minimizes the number of defects. An FDT computer may be fabricated with a technology with a much higher

defect rate, and should be able to operate effectively with one or more hardware defects. This can be done at the circuit, module, architectural, or software level.

Operational reliability is equally important. The FDT computer must be able to continue to operate in the presence of operational hardware failures. The FDT computer must be able to mask faults so they do not impact software operation, or to detect the effects of the faults and recover from them at a minimal impact to the application. In addition to hardware failures, soft errors and SEUs should be detectable and correctable.

These three goals must be balanced against the requirements of testability, power, cost, and runtime performance. It might be easy to improve reliability using very large amounts of redundant hardware. In practice, however, this increases the size of the layout, manufacturing cost, and power consumption. It is also likely to increase propagation delays and reduce the speed of the processor. Thus, trade-offs must be made between the factors to achieve the best yield and reliability in the most efficient manner.

The FDT computer is compared to conventional computers using the same criteria used today. Application benchmark performance, purchase price, power consumption, and reliability continue to be important. For a FDT computer to be practical, it must compete with conventional CMOS processors in all of these areas. Along with reliability comes ease of use. Fault and defect tolerance techniques should be transparent to the user, performed at the hardware and operating system levels. Reliability is a key concern, and customer confidence is critical to the widespread adoption of non-CMOS technologies. For a long time, CMOS and its successor are likely to co-exist and compete. It will take time for the customer community to accept the idea of circuits that are not completely defect free. While there is some precedent that shows customers will buy products with defects (e.g., memory chips), the idea of a CPU with an unknown number of device defects may require a shift in paradigm.

7.3 *Fault Tolerance Strategies*

A FDT computer incorporates fault tolerance at several levels, improving overall system reliability at each step until an acceptable result is achieved. Table 7.1 summarizes techniques and the level of abstraction for which it is best suited.

7.3.1 Module/Circuit Level. The majority of fault tolerance capability will likely be at the module or circuit level. In conventional CMOS technologies, manufacturing testing removes chips with hardware faults prior to operational use. Operational hardware faults are rare enough that replacement of large modules is feasible. Soft errors and SEUs can often be handled at the operating system or application level. In a future device technology, however, faults and errors will be more common, and must be dealt with at lower levels.

7.3.1.1 Fault and Error Detection. The FDT processor will detect faults at the module and circuit level using the following methods:

Duplication With Comparison/Concurrent Error Detection(DWC/CED).

Implemented at the level of small hardware modules, DWC and CED perform an operation in parallel and compare results. Soft errors are detected easily, as well as many hardware faults in either module. If a fault is detected, a hardware exception is raised to allow the operating system to diagnose and recover from the fault.

Error Correcting Codes. Often used to mask the effect of SEUs and faults, ECC in memories can also be used to detect faults if the decoder signals an exception when an error is detected.

Localized BIST. BIST can be done at several levels, either under local control or at the chip level. Localized BIST may be useful in that it requires minimal long distance interconnect, can be run quickly, and can perform repairs without knowledge or intervention of the higher levels.

Table 7.1: A fault and defect tolerant processor will likely use many different fault tolerance techniques at several levels of abstraction. Low level techniques are best suited for permanent faults detection and recovery.

Technique	Module	Instruction	Operating System	Application
DWC	HW/SW Detection	HW/SW Detection		SW Detection
CED/RWSO	HW/SW Detection			
ECC	HW/SW Masking	HW/SW Masking	HW/SW Masking	
Timing Checks	HW detection			
RMR, VNM, TMR-R, Reconfig	HW/SW Masking			
Spare Rows/Cols	HW/SW Repair			
Pipeline register ECC	HW/SW confinement	HW/SW confinement		
Local BISTR	HW detection, diagnosis, repair			
SIR		SW detection, recovery		
EXE unit replication		HW/SW detection		
BIST Instructions		HW detection, diagnosis		
BISR Instructions		HW repair		
Memory scrubbing			SW repair	
Memory Address Range Checking			HW/SW detection	
SEU Rate Monitoring			SW diagnosis/repair	
SW Replacement			HW/SW repair	
OS Cache control			HW repair	
OS BIST			HW diagnosis	
OS BISR			HW repair	
Checkpointing			HW/SW recovery	
Reasonableness Checks				SW detection
Reversal Checks				SW detection
Modular Coding				SW recovery

Timing checks. Used to detect parametric faults (e.g., faults causing device switching speeds to be slower than required). This will likely be performed by local BIST, with faults being signalled by raising an exception. Recovery would typically involve shutting down the faulty module and replacing it with a spare.

7.3.1.2 Masking. Fault masking is extremely important, as repair and recovery under the explicit control of a higher level is often difficult and diminishes performance. Several hardware fault masking techniques that will be useful at the module level include:

R-Modular Redundancy/ Tri-Modular Redundancy. Conventional TMR and RMR performed at the module level. The benefit of RMR depends on the size of the hardware module and the number of outputs.

Modular Reconfiguration. As discussed in Section 5.3.3, reconfiguration can provide benefits over RMR when soft errors are infrequent. Increasing the number of redundant modules generally increases FT performance. Modular reconfiguration is done with larger modules, and fewer choices on interconnection than an FPGA-like architecture, but requires significantly less overhead.

TMR-protected Reconfiguration. Combines the benefits of RMR and modular reconfiguration. Multiple modules are implemented in hardware, and are connected to a majority voter, which provides protection from soft errors.

Spare rows and columns. Spare rows and columns are modular reconfiguration as applied to the memory structures.

Error Correcting Codes. ECC is very useful in memories to mask the effect of SEUs. It can also protect control signals that propagate through the pipeline stages in pipeline registers until they reach the intended stage. Fast codes are important to minimize latency.

FPGA-like reconfiguration. FPGA-like architectures provide significant reliability benefits due to the fine granularity in configurability. However, there is a

significant impact on speed and hardware cost due to the overhead of redundant interconnect and configuration hardware. Provided that suitable yield and reliability can be obtained through coarse-grain reconfiguration (such as modular reconfiguration) and other fault tolerance techniques, FPGA-like structures are not likely to be adopted.

7.3.1.3 Diagnosis, Repair, and Recovery. Localized BIST/R can determine which module is faulty. When possible, repair should be done without requiring operating system involvement. The pipeline can be stalled while diagnosis is performed, provided that testing can be done quickly. If the fault is not in a critical part of the hardware (e.g., a fault in a floating point adder impacts floating point instructions but would not require integer instructions to be halted), diagnosis and repair can be handled at the operating system level. In this way, unrelated operations can continue.

7.3.2 Instruction Level. Instruction level techniques include assembly language instructions to perform FT functions as well as hardware support at the architectural level of the CPU (e.g., FT support in the dynamic instruction scheduling hardware or cache control logic).

7.3.2.1 Detection. Duplication with comparison (DWC) can be used at the instruction level by having the pipeline scheduler issue the same instruction twice and compare the results prior to the instruction commit phase. This is useful for detecting soft errors lasting only a single clock cycle. Instruction DWC would not detect hard faults, since the same result would be returned both times. However, if the processor has multiple functional units (e.g., integer adders), the instruction can be executed on different functional units to detect hard faults. Of course, the design of the instruction scheduler becomes more complex.

Single Instruction Retry (SIR) is a similar technique but is called by exception and used in conjunction with ECC. If an error is detected by an ECC decoder in a

pipeline register, the instruction can be flushed prior to committing the results to the register file or to the cache. The scheduler would then reissue the instruction. If the error was a soft error, the result would be correct on the second iteration. If the error recurs, an exception can be raised and fault diagnosis and repair handled at a higher level.

7.3.2.2 Masking. ECC can be used at the instruction level. Modern processors are typically 64-bit architectures. In many cases, 64 bits are not required (e.g., single precision floating point numbers require only 32 bits, and ASCII *char* variables use only eight bits). ECC protected instructions could be created to protect data in a manner transparent to the higher levels. For example, the 64 bit data word can be encoded and stored in the cache. It would be decoded prior to arithmetic operations and recoded prior to results storage. If a *systematic* code is used, the uncoded word remains unchanged as part of the code word. Thus, operations can be performed on just those bits, eliminating the need to decode the word at each step. Updated code bits are recomputed only when necessary as determined by the processor or the operating system.

7.3.2.3 Diagnosis. The instruction set may include special BIST instructions to assist in the test process. These instructions may be used by the operating system to access internal registers or force connections to specific hardware modules so that testing can be controlled.

The instruction set may also be augmented so instructions can be executed on particular functional units in the ALU. This method of *EXE unit replication* can be used by the operating system to rerun instructions on two or more functional units so the results can be compared.

7.3.2.4 Repair. In addition to BIST instructions, the processor may implement instructions for BISR. These instructions would be used by the operating system to control modular reconfiguration, disconnecting faulty modules throughout

the pipeline and connecting functional spares. The advantage of BIST and BISR instructions is a conventional datapath can be used without adding a large number of additional wires used only for testing.

The instruction scheduler can be used to repair faulty modules. It is possible to develop a *fault tolerant Tomasulo scheduler*. If the processor possesses multiple functional units of each type (e.g., multiple floating point adders, integer dividers, etc.), the scheduler can be made aware of functional status of each unit, avoiding disabled units. Thus, one architecture can be fabricated. Redundant hardware contributes to application performance since the scheduler can issue instructions to these functional units. At the same time, performance degrades gracefully if one or more units are defective. Following yield testing, a processor with more operational units can be sold as a higher performance processor versus one with fewer operational functional units.

7.3.3 Operating System Level. Many of the fault tolerance techniques introduced at the lower levels require complicated control that is best implemented by the operating system. Errors will typically trigger an exception which would invoke the operating system BIST and BISR routines.

7.3.3.1 Detection. One method of error detection is *memory address range checking*. The operating system could check memory addresses to ensure they are in a valid range for the program (e.g., in the application memory space and not the operating system protected memory). Invalid addresses can be the result of application software error, malicious code, soft error, or hardware fault in the processor. The latter two causes can trigger the operating system to run BIST/R routines to determine the health of the processor and repair faulty modules.

7.3.3.2 Masking. Cache preloading is a task sometimes handled by the operating system. This process can be extended to handle faults in the cache memory. *False* and *conflicting hits* are described in Chapter VIII. These events require one or more cache lines to be invalidated and reloaded from main memory.

If the device technology is subject to high rates of permanent operational failures (i.e., hard faults), the cache capacity should gracefully degrade as defects occur. The operating system can be used to monitor cache size, changing its preloading strategy as appropriate. Since the operating system is aware of the fault status of each cache row, it can avoid faulty lines. This will reduce the complexity of the cache control at the cost of performance, since the work is now done in software.

The operating system can also implement ECC. If not provided at the hardware level, the operating system can provide ECC protection of application data transparently. One advantage of this technique is the code choice can be changed based on the soft error rate in the environment. If the hardware provides a soft error rate monitor, the operating system can switch ECC codes to provide a level of protection that matches the need. If the soft error rate is low, ECC may be turned off to maximize performance. If the soft error rate is high, complicated ECC codes can be used to provide better soft error protection (especially for critical areas of the operating system), at the cost of slower performance.

7.3.3.3 Confinement. Operating system level checkpointing can isolate errors that occur in a particular module of code.

7.3.3.4 Diagnosis. Many of the techniques proposed require the operating system to control testing and recovery. The operating system should include BIST routines to isolate faults to a particular module or hardware unit. These routines can be triggered by exception when errors are detected at lower levels, or periodically by the operating system itself. The BIST routines run during idle cycles to minimize impact to the application code.

7.3.3.5 Repair. The operating system can repair defects in several ways:

BISR routines. The operating system can include the routines to reconfigure or disable faulty modules.

Memory scrubbing. Errors will accumulate in ECC words stored in memory due to SEUs and hardware faults. The operating system or cache controller should periodically scrub the memory contents to restore correct values.

Reconfiguration control. The operating system can control the operation of the low level hardware protected by modular reconfiguration and FPGA-like reconfiguration. Faulty modules are disabled, and replacement modules are turned on and connected.

Dynamic routing. If a fine-grained reconfigurable architecture is used in the CPU, the operating system can be used to reconfigure the design around faulty components. Dynamic routing is very expensive in terms of performance, but could provide very high reliability.

Software replacement. Software routines can replace faulty hardware modules. For example, software versions of floating point routines could be substituted by the operating system to replace faulty hardware units. The processor could raise an exception when an executing instruction requires a failed hardware unit, triggering the operating system to call the software version.

7.3.4 Application Level. Application level fault tolerance can detect and recover from soft errors and hardware faults as well. Detection can be done through the use of “reasonableness checks” and “reversal checks” as described in Chapter II. Confinement and recovery techniques can be implemented using modular coding and checkpointing. Upon detecting an error, the application calls operating system routines to diagnose and recovery from the error. As a general rule, however, the FDT processor should not rely on the application code to detect or recover from faults.

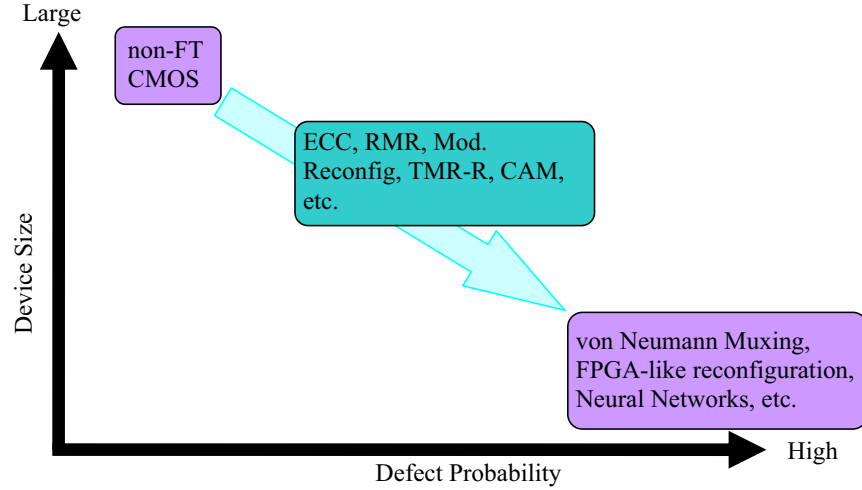


Figure 7.1: Fault and defect tolerant computing is likely to follow an evolutionary path. Techniques with moderate redundancy requirements are preferred, provided the device technology can be made suitably reliable. In the long term, extremely unreliable technologies may be adopted but will require very aggressive fault tolerance techniques such as von Neumann multiplexing.

7.4 Proposed Architecture

The evolution of computer architecture to adapt to new device technologies will be a gradual process. While nonclassical architectures such as quantum computers and neural nets may one day replace the von Neumann architecture, it is likely fault and defect tolerant computer will be an adaptation of a modern conventional architecture. One or more of the fault tolerance approaches described in this chapter will be incorporated into a traditional design. Ideally, the fault tolerance capability is transparent to the application developer. The choice of techniques and the level of fault tolerance will largely depend on the requirements of the device technology. The most successful future technology will likely be the one that requires the least modification of a conventional architecture.

While von Neumann multiplexing was shown in Chapter VI to be highly effective, it requires large amounts of redundant hardware. To compete with CMOS, a future device technology will have to be several orders of magnitude smaller than

CMOS. Similarly, fine-grain reconfiguration is very effective, but at a large cost in redundant hardware, as well as the performance cost of dynamic routing.

For these reasons, the remainder of this document will focus on a “mid-term” architecture. While incorporating much more aggressive fault tolerance techniques than modern CMOS-based designs, the architecture does not require the very large levels of redundancy that are required to implement von Neumann multiplexing or other techniques. The following chapters show that RMR, modular reconfiguration, TMR-R, and other moderately expensive fault tolerance techniques can implement a processor using devices with defect rates higher than 10^{-6} , three orders of magnitude higher than conventional CMOS. Beyond this point, the most aggressive techniques are required.

The proposed fault and defect tolerant processor architecture is a 32-bit pipelined RISC architecture, adapted from the classical MIPS design [PH98]. The top level architecture is shown in Figure 7.2. The design has five pipeline stages, with separate instruction and data caches. While not shown in the figure, the proposed design does incorporate a full IEEE-754 compliant floating point unit. The architecture of the unit is adapted from [MP00].

The general design approach maximizes the yield of each major module in the processor through hardware fault tolerance techniques. Since the yield of the overall CPU is bounded above by the yield of each module, care is taken to examine each module in the processor.

While all of the fault tolerance methods described in this chapter may be adopted, the most important techniques are those at the module and circuit level. Given the relative frequency of defects, the probability any module will function without circuit level fault tolerance is very low. Higher level fault tolerance only becomes effective when some minimum level of reliability can be provided by the lower level fault tolerance techniques. Thus, the design incorporates most of the module level techniques. Most instruction, operating system, and application level fault tolerance

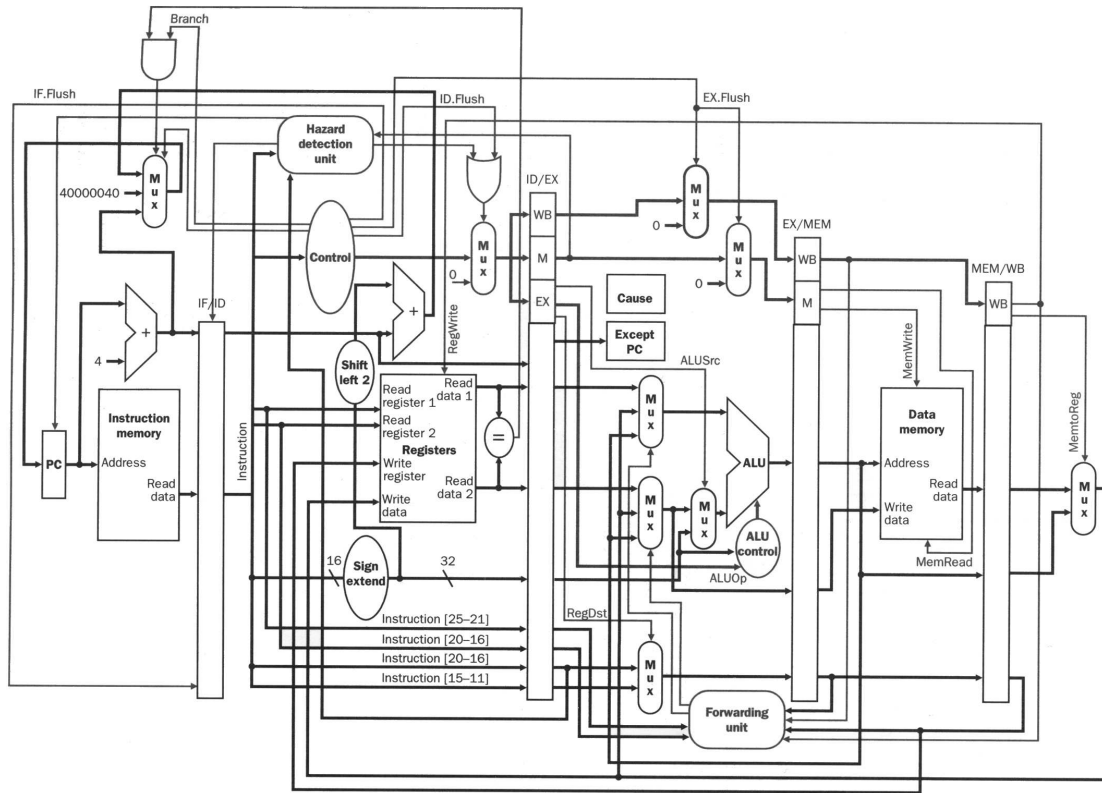


Figure 7.2: The proposed fault and defect tolerant processor architecture is based on the classic single-issue MIPS 32 bit pipelined RISC design [PH98]. While included in the FDT design, this diagram does not show the floating point unit.

techniques are described notionally, but will not be modelled further in subsequent chapters.

A minimal FDT design must incorporate the following techniques:

Module and circuit level:

- ECC
- Timing checks
- RMR/TMR
- TMR-protected reconfiguration
- Modular reconfiguration (including spare rows)
- Localized BIST/R
- Fault detection exceptions

Instruction and Operating System Level:

- SEU Rate monitoring
- Software replacement of Hardware functions
- Operating System BIST/R routines

The conceptual operation of the FDT processor is described in the next section.

7.5 Concept of Operation

This section covers four aspects of the FDT processor: yield testing, startup configuration, runtime testing, and soft error detection and recovery.

7.5.1 Yield Testing. At time of fabrication, each chip must be tested to ensure some minimal level of functionality. With an unreliable nanotechnology, defects are highly likely to be present in each processor. Testing determines the location and effect of the defects, and whether the fault tolerance built into the architecture can

correct them. Chips that are unable to perform the required function are discarded. Depending on where the defects occur, it may be possible to provide a diminished level of functionality (e.g., reduced cache size or an EXE unit with fewer functional units). These chips would be sorted and sold separately.

VLSI testing is a complicated field, and the specific testing methods are beyond the scope of this research. Notionally, test techniques for conventional architectures can be used to test the FDT processor. A *design for testability* approach will be used to simplify the testing process by incorporating special test hardware into the architecture, such as JTAG hardware, localized BIST/R units, and boundary scan shift registers.

The test process must examine each module in the processor to determine functionality. Functional units protected by RMR, TMR, modular reconfiguration, and TMR-R are correctable if some minimum number of modules are fault-free (e.g., two for TMR). Larger modules such as the cache memory are functional if the number of faulty rows does not exceed the number of spares. Inevitably, some of the logic in the processor will not be protected (i.e., the “chip kill logic”). For example, a fault in the majority voter in a TMR module will likely disable the entire functional unit, even if all three redundant modules are functional. The test strategy must, at a minimum, examine all the chip kill logic in the processor.

During the test process, known defects should be mapped to create an initial configuration for all modules protected by modular reconfiguration or TMR-R.

7.5.2 Startup Configuration. The processor contains a number of modules protected by modular reconfiguration and TMR-R. If the technology supports nonvolatile memory storage, the settings for these modules can be programmed during yield testing and modified only when new faults are determined. However, if nonvolatile storage is not available, the configuration settings must be loaded upon startup in a manner similar to the modern FPGA.

Startup configuration should be controlled by hardware with a very high reliability. The FDT processor is likely to have a number of defects and is unlikely to be able to configure itself directly. A configuration loader should be implemented either on the same chip as the processor, or on a separate chip. In either case, the configuration loader should be implemented using highly reliable technology. If the device technology supports it, reliability of the loader can be increased by increasing the device size. If the reliability cannot be increased using the same technology as the main processor, a separate configuration loader can be implemented in silicon CMOS.

The most straightforward approach to programming the configuration registers is to use a long shift register, similar to that used in FPGAs. Each of the configuration registers in the functional units protected by modular reconfiguration is connected in series. In effect, the FDT processor is actually a very coarse-grained FPGA. The number of configuration bits will be much less than an FPGA, however, and configuration time will be very short.

7.5.3 Runtime Testing. Runtime testing is important for two reasons: the device technology may be subject to operational hard failures, and yield testing may not locate all defects present at manufacture. Runtime testing allows the processor to detect these errors, repair them when possible, and continue operation.

Runtime testing should be performed periodically during processor operation. Test routines can be run at the local module level, and at the instruction level. The test instructions would be stored in ROM, similar to the Power On Self Test (POST) routines in modern computers. Localized testing could be done in parallel, with each unit (e.g., the instruction cache, data cache, ALU, etc) testing itself. Instruction level test would augment localized testing for those circuits not covered by local BIST.

If possible, the local BIST/R units will reconfigure their spares internally to repair detected faults. If correction is not possible, an exception would be immediately signalled. The operating system would then attempt to repair the fault. A fault may be severe enough to render the pipeline inoperative (e.g., a fault in the majority voter

providing TMR to the program counter incrementer). If the pipeline is inoperative, a fault signal is output from the chip. While the effects are severe, the number of devices in these critical portions of the chip are relatively small. For the instruction cache, the number of critical devices is approximately 15,000 (cf., Chapter VIII).

7.5.4 Soft Error Recovery. Even if the rate of operational hardware faults is low enough to reduce the need for runtime testing, soft errors and SEUs will be common. The FDT processor must detect and correct these errors. Many effects are masked by RMR and TMR-R, reducing the probability that incorrect results will be latched into the pipeline registers and committed to the registers or cache. ECC will mask the effects of SEUs that occur in the registers and cache.

The processor should monitor the rate of soft errors and single event upsets. As the rate of SEUs increases, the operating system should scrub the cache memory more often. Error correcting codes can protect against both hard and soft errors. Frequent scrubbing increases the number of hardware faults that can be masked.

7.5.5 The Role of the Operating System. The operating system performs the following functions:

- SEU rate monitoring
- SEU scrubbing
- Software replacement of hardware functions
- Operating system BIST/R routines

The operating system controls processor function and coordinates the activities of the local BIST/R modules. It also reduces the hardware complexity of the BIST/R modules by performing some of the work in hardware. In addition, it can provide a software backup for the hardware BIST/R functions. In many cases, even if the localized BIST/R module fails, the pipeline is still functional and the operating system can override the hardware BIST/R.

The operating system also provides routines to replace the floating point functions if the hardware units are faulty. Failures in the floating point unit do not effect pipeline operation, and are easily recoverable through software emulation, albeit at a much longer execution time.

7.6 Conclusion

This chapter has outlined the high level architecture of the fault and defect tolerant processor. It incorporates fault tolerance techniques at the modular, instruction, and operating system levels to increase yield and operational reliability for both hard faults and soft errors. A notional concept of operations described the fabrication and runtime testing, as well as how defects would be detected and recovered. The next two chapters discuss the implementation details for the CPU, and show that an acceptable yield can be achieved even with devices with failure rates as high as 10^{-6} . As more than 90% of the devices in a modern microprocessor are in the cache memory, it is examined first. Chapter VIII proposes an effective Content-Addressable Memory design. The remainder of the CPU is examined in Chapter IX, and overall chip yield is determined.

VIII. Cache Memory

8.1 Introduction

In a modern microprocessor, register and cache memory accounts for as much as 90% of the transistors on the die. Thus, memory reliability is a key factor in the reliability of the overall processor. Fault and defect tolerance techniques are already widely used to improve the yield of silicon CMOS memory chips. Spare columns are used to increase manufacturing yield, while ECC is becoming common for soft error tolerance. Indeed, Microsoft recently proposed the use of Error-Correcting Code-protected RAM for general purpose computers running the Windows Vista operating system [Cou06].

Most fault tolerant memory designs use fairly conservative fault tolerance techniques. Only moderate levels of redundancy are needed to overcome device defect rates in the range of 10^{-9} to 10^{-7} . More aggressive techniques such as von Neumann Multiplexing may be required to use future nanodevices with defect rates that may be orders of magnitude higher.

This chapter develops the functional architecture of the cache memory in the FDT processor, meeting the objectives of goal two. It examines the problem of creating a reliable microprocessor cache memory architecture using nanodevices with defect probabilities in the range (10^{-7} to 10^{-3}), focusing on the architectural level. Thus, Boolean logic is used and reliability-improving gate design optimization is not required. A novel fault tolerant Content Addressable Memory (CAM)-based architecture is proposed and analyzed for effectiveness versus a non-fault tolerant CAM as well as a conventional cache architecture. While most previous work examines the memory array in isolation, this research incorporates support and control circuitry to create a more accurate model of the overall memory yield. Requiring approximately 15 times more devices than the conventional design, the new design is well suited for future nanotechnologies. Simulation shows a 90% yield for process technologies with defect rates as high as 4×10^{-6} , three orders of magnitude higher than non-fault tolerant designs.

Table 8.1: Cache Characteristics

Cache Size	64KB	512KB	1MB
Block Size (Bytes)	64	64	64
Cache Lines (per bank)	512	4096	8192
Tag Bits per block	46	43	42
Total Data Bits	524,288	4,194,304	8,388,608
Total Tag Bits	47,104	352,256	688,128
Total Size (KB)	70	557	1112

8.2 Background

8.2.1 Cache Characteristics. Three cache sizes are considered: 64KB, 512KB, and 1MB. These sizes are representative of modern microprocessors. The techniques described in this chapter are easily scalable to larger cache sizes. Each cache is two-way set-associative and dual-ported. A summary of the cache characteristics is shown in Table 8.1.

A block diagram of a typical memory is shown in Figure 8.1. The memory cell array contains the largest number of devices, and has been the focus of much attention to improve reliability. The address decoders, output buffers, access and timing control circuits, Error Checking and Correction (ECC), and Built-In Self-Test/Self-Repair (BIST/R) hardware consists of predominantly combinational logic.

8.2.2 Yield Modelling. The mathematical models used in this analysis are found in Chapter V. Basic yield expressions are introduced in Section 5.1.1. Fault types are discussed in Section 5.2. Clustering of defects is discussed in Section 5.1.2. Yield of modules formed from multiple components is discussed in Section 5.1.3.

8.2.3 Hardware Fault Tolerance Techniques. The cache design relies primarily on moderately redundant fault tolerance techniques (i.e., TMR, modular re-configuration, TMR-R, and ECC). Models for the basic fault tolerance techniques are discussed in Section 5.3. Memory fault tolerance is discussed in detail in Sections 5.4.2 and 5.4.3.

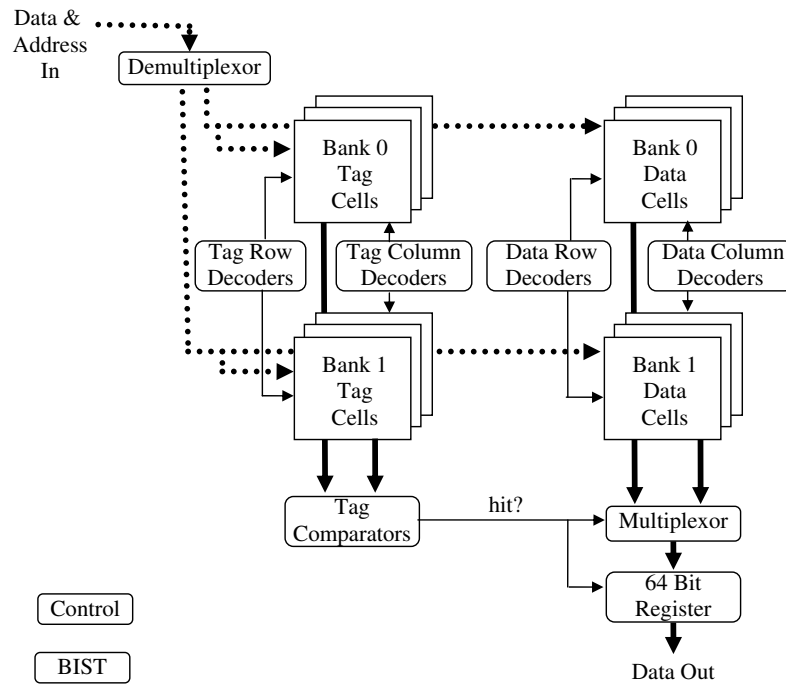


Figure 8.1: A typical Single Read Port Cache. The write path is shown as a dotted line. The read path is shown in bold. Dual-ported caches replicate the tag comparators, multiplexer, and output registers at the bottom of the figure.

Von Neumann multiplexing [vN56] was considered for the FDT cache design. However, it requires large amounts of redundant devices in parallel and is not generally beneficial unless the entire logic chain can be replicated in parallel. Von Neumann multiplexing can protect large logic modules, combining the parallel result into a single output signal with a majority gate. However, the reliability of this technique is limited by the reliability of the majority gate. Given the large number of devices in the cache, use of von Neumann multiplexing is not economical for device technologies with $\lambda_1 < 10^{-5}$. This chapter and Chapter IX show less expensive techniques are capable of achieving high yields when $\lambda_1 < 10^{-5}$.

Threshold logic gate (TLG) circuits have also received some attention for fault tolerance and neural networks applications [LC67, Rei00]. Theoretically, TLG circuits can be made arbitrarily fault-tolerant using small to moderate amounts of redundant hardware, while Boolean circuits cannot [Rei00]. Threshold logic gates can be made to implement any Boolean function and could replace conventional Boolean gates. However, threshold logic design differs greatly from Boolean design, and new design and synthesis tools will be required [BQA03]. For the near to mid-term, computers will continue to be constructed from Boolean logic gates.

8.2.4 Previous Fault Tolerant Memories. Several fault tolerant memory designs have been proposed [Lo94, SSS02, CLM⁺03], including standard memory and content-addressable designs. These designs typically target modern CMOS, which has a relatively moderate defect probability.

A variety of memory designs have been proposed that combine several fault tolerance techniques into a single architecture. These designs typically use a combination of spares and ECC to provide both manufacturing defect tolerance and runtime SEU fault tolerance. A synergistic effect of this approach was first observed in [SK92]. In that design, spare rows/columns were combined with ECC to result in yields that were higher than either technique alone.

8.2.5 Error Correcting Codes. Forward error correction is increasingly used in both DRAM memories as well as in modern CPU caches. Error Correcting Codes (ECC) can correct both hard faults and Single Event Upsets (SEUs). A large number of codes exist, but those most useful can be implemented quickly and efficiently in hardware. Simple codes, such as the (72,64) Hamming code, are used in the AMD Athlon 64 [Adv05]. These codes can detect two errors in the 72 bit codeword and correct any one error. Another class of codes are the simple Hamming codes, such as the (7,4) code which can correct one error. Using slightly more hardware, the (24,12) Extended Golay code can correct any three errors [LDJC83].

The use of ECC should not have a large impact on cache latency for small codes, as there are fast parallel implementations of both the encoders and decoders for extended Golay codes. One parallel implementation of the decoder requires only 617 logic gates (roughly 2400 devices) [BMH00]. A parallel encoder is easily designed using the algorithm in [LDJC83] and requires 1744 devices. Memory ECC is discussed further in Section 5.4.1.

8.2.6 Support Logic Fault Tolerance. When predicting the yield and reliability of memories, it is equally important to consider the support modules in addition to the memory array. From (5.11), it is evident the yield of the overall cache or chip is bounded above by the yield of any individual module. The benefit of a highly redundant and reliable memory array can be sabotaged by a control module with no fault tolerance. Thus, it is equally important to model the performance of the control logic, Built-In Self-Test (BIST) module, multiplexers, registers, and other hardware in the cache. Memory elements such as registers can be protected in a manner similar to the main memory array. If the device technology allows, support circuits can be implemented with more reliable devices (e.g., by making these transistors physically larger in the VLSI layout). If this is not possible, other fault tolerance techniques can be used. The three most common techniques for combinational logic are *R-Modular Redundancy*, *Reconfiguration*, and *Multiplexing*. Yield models for RMR are found in

Section 5.3.2; modular reconfiguration is in Section 5.3.3; and TMR-protected modular reconfiguration is in Section 5.3.4.

8.2.7 Assumptions. As discussed in Chapter IV, several assumptions are necessary prior to analysis. First, all devices are identical (i.e., reliability of key devices is not increased by altering their size or physical characteristics). Second, majority gates are implemented as simple Boolean circuits. Third, faults are modelled such that a fault in any device in a module disables that module. Finally, two simple defect clustering models are used: a non-clustered model (i.e., Poisson distribution), and a large-scale clustered model (i.e., the negative binomial model [Kor89].)

8.3 Architectures for Comparison

This section analyzes the fault tolerance performance of two cache architectures intended for low defect rate technologies. The key comparison parameter is *Maximum Allowable Defect Probability* (MADP). MADP is defined as the maximum defect probability of the devices used in the cache, such that an acceptable manufacturing yields and runtime reliability is achieved. This section first examines a typical cache incorporating no fault tolerance. Next, a typical CAM-based cache is analyzed whose only fault tolerance is row sparing. Later in this chapter, an improved fault tolerant CAM-based cache architecture is proposed and shown to possess a greater MADP.

8.3.1 Cache A: Non Fault Tolerant Fixed Design. This design is representative of a typical cache designed to implement maximum memory capacity with the smallest number of devices. The required modules forming the cache are summarized in Table 8.2. For each type, the number of modules required, as well as the number of devices (i.e., transistors) is shown for each of the three cache sizes. In general, the cache is made up of four memory modules, two each for tag and data. Each memory module can be viewed as a three-dimensional block. The row and column are determined by the address lines. For any address, 64 data bits are returned in parallel and form the Z dimension. Thus, the size and numbers of the row and column

Table 8.2: Cache A (No Fault Tolerance) Components

	Mods	Number of Devices per Mod		
Cache Size		64KB	512KB	1MB
Control	1	100K	100K	100K
Data Mem Array	2	2.10M	16.8M	33.6M
Tag Mem Array	2	208.9K	1.57M	3.08M
Data Row Decoders	6	910	4,626	4,626
Data Col Decoders	6	910	2,064	4,626
Tag Row Decoders	6	396	910	2,064
Tag Col Decoders	6	170	910	910
Tag Comparators	4	882	828	810
2-to-1 by 64 Muxes	2	770	770	770
64 bit registers	3	2,178	2,178	2,178
1-to-2 by 64 Demux	1	514	514	514
N_{cache}		4.74M	36.86M	73.45M

decoders can be easily determined. For example, the 64KB cache requires two data memory banks, each of 32KB. Each memory bank is a 64 by 64 by 64 array. Each read and write port in each bank requires its own row and column decoders. Since a set-associative cache has two data banks, the total number of row decoders is thus 6. The larger caches require more complex decoders.

Since there is no redundancy, all N_{cache} transistors must function for the cache to be functional. Yield is therefore (5.5) for the unclustered case, and (5.10) for the clustered case. The number of transistors in the control module is an estimate, but has little impact on overall cache yield. Yield is dominated by the large number of devices in the memory arrays. Unless $N_{control}$ exceeds 10^6 devices, it does not have significant impact on yield.

The unclustered yield of the three caches is shown in Figure 8.2. Since the yield of the entire processor can be no greater than the yield of the cache module, the region of concern is above $Y_{min} = 0.9$. In Chapter IX, the entire CPU is considered, and the target yield becomes $Y_{min} = 0.7$. From the plot, it is evident that even small caches possess a MADP of no more than $\lambda_1 = 10^{-7}$.

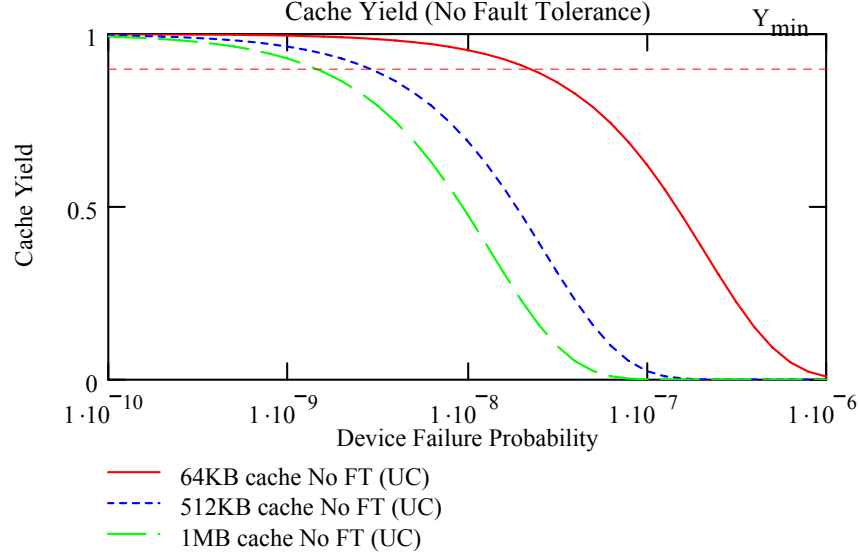


Figure 8.2: Predicted unclustered (Poisson) yield for no fault tolerance. $Y_{min} = 0.9$.

Table 8.3: Cache A (No Fault Tolerance) Maximum Allowable Defect Probability

Cache Size	64KB	512KB	1MB
N_{cache}	4.74M	36.86M	73.45M
MADP (unclustered)	2.22×10^{-8}	2.87×10^{-9}	1.44×10^{-9}
MADP (clustered)	2.53×10^{-8}	3.16×10^{-9}	1.60×10^{-9}

Yield increases slightly when defect clustering is considered. For the 64KB cache, these effects are shown in Figure 8.3. However, the improvement is minimal at the top end of the curves where $Y > 0.9$. To determine MADP in practical cache design, defect clustering can be ignored. Monte Carlo simulations results for Cache A are summarized in Table 8.3.

8.3.2 Cache B: Basic CAM Design.

8.3.2.1 Design and Operation. This section describes the design of a Content-Addressable Memory (CAM)-based cache architecture, models its yield, and identifies the limiting performance factors. Figure 8.4 shows the basic architecture. The implementation is fully-associative, and thus a cache word of 64 bytes can be

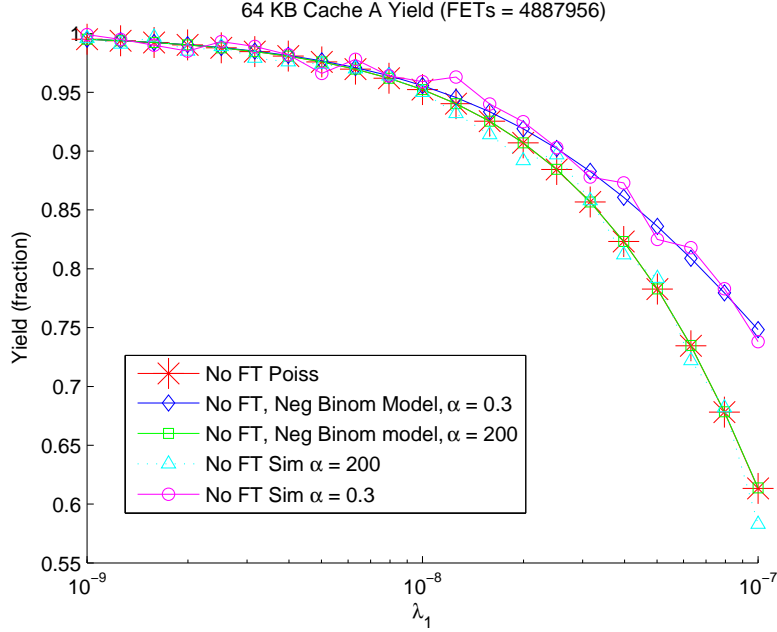


Figure 8.3: Simulation results for unclustered (Poisson) and clustered (Neg Binomial) yield for no fault tolerance. $Y_{min} = 0.9$.

placed in any of the CAM entries. Depending on the cache size, the CAM requires 1024, 8192, or 16384 entries, not including additional entries used as spares.

The cache has two read ports. To perform a read operation, the read address is placed on either the $A1$ or $A0$ address bus. Each of the CAM words compares the addresses on this bus with the address of their stored data. If the addresses match, the data register is connected to the *Data Out (DO)* bus. The entire 512 bit cache block is passed to a multiplexer, which selects the correct 64 bit word to pass out of the cache.

The control module of the cache consists of three parts: a *CAM Status Register (CSR)*, the *Next CAM Word Register (NCWR)*, and control logic. The CSR uses one bit per CAM word to store the fault status. Upon startup and periodically during operation, the Built-In Self-Test/Repair (BIST/R) module tests the status of each CAM word. Faulty words are removed from use and replaced with the spare words in the CAM array. The NCWR determines which CAM word to use for the next

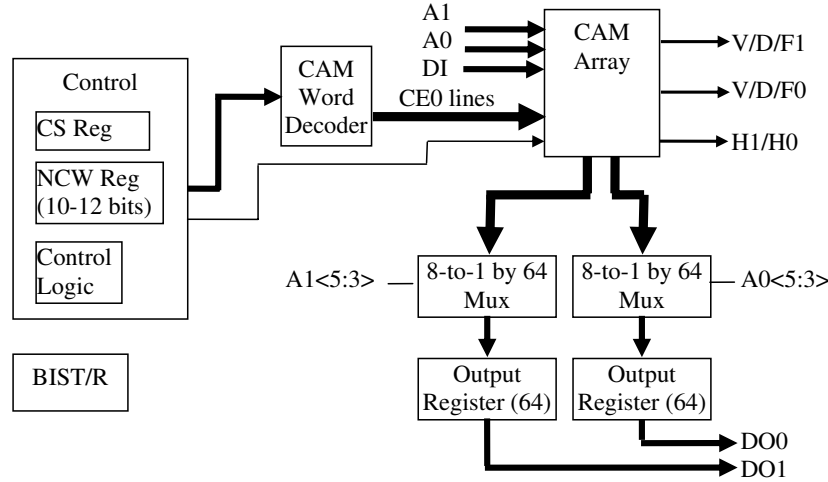


Figure 8.4: Simple CAM cache with dual read ports and one write port (Cache B). Some control signals omitted for clarity.

cache write operation. The cache uses a modified least-recently used strategy for block replacement. The control logic block includes the remaining combinational and sequential logic to manage cache operation.

The final two blocks in the cache are the CAM Word Decoder and the BIST/R module. The CAM Word decoder is a 10-to-1024 bit address decoder that selects a CAM cell for writing. The BIST/R performs initial testing to determine the fault status of each of the CAM words. It cycles through the CAM words, testing the registers, comparators, and output transmission gates for correct operation. Fault status is stored both in a register in each CAM word as well as the CSR in the control module.

Figure 8.5 shows the design of a single CAM word. The CAM word consists of registers to store the tag and data fields, as well as word status (e.g., valid and dirty bits, plus an additional bit to store the fault status of the CAM word). Some signals are omitted for clarity. Since the word incorporates no fault tolerance circuitry, a single faulty device will result in the entire CAM word being labelled as faulty.

Transmission gates connect the CAM word to a common output bus. To limit the number of devices in the CAM word, all 512 bits in the CAM data word are

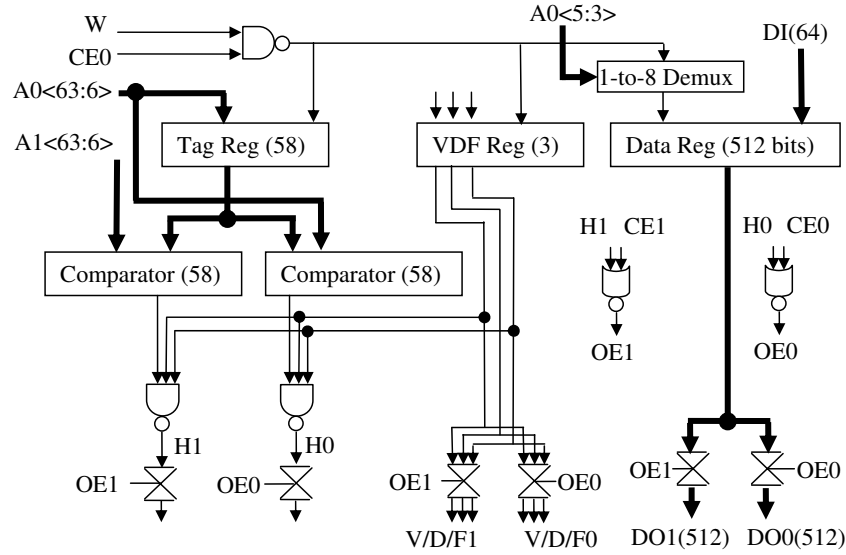


Figure 8.5: Simple CAM Word design for Cache B. Dual read ports are shown. Only one write port exists. Data is returned in 512 bit cache words. Writing is done in 64 bit words

returned on the appropriate data bus when a hit occurs. A key limitation of this approach is that a fault in one of these transmission gates may disable the entire CAM array.

All of the CAM words are connected to the data buses, but in normal operation no more than one word will ever match a requested address except in the case of Single Event Upset (SEU). If two or more words do match, a *conflicting hit* occurs. One limitation of this particular architecture is the control unit cannot determine which CAM words are returning the hit. For a conflicting hit, the control unit must invalidate the entire cache, retest to determine if any of the CAM words have permanent faults, and reload the cache. A more serious problem is the possibility of a *false hit*, in which a SEU flips a bit in the tag memory such that a single row incorrectly matches a read request. False hits are undetectable in this architecture. The FDT cache architecture (developed later) corrects both problems. It also reduces the probability of false hits, and includes circuits to isolate the CAM words generating false and conflicting hits, allowing only the affected words to be invalidated.

Finally, the 1-to-8 demultiplexer is used for cache writes. The memory architecture uses 64-bit words. Thus, a single cache write will change only 64 bits in the 512 bit CAM word. The demultiplexer enables the Write Enable lines for the proper 64 register flip-flops.

The cache can be either write-back or write-through. In environments subject to SEUs, a write-through architecture is preferred. If a SEU changes a bit in a tag address stored in the cache, a requested cache block may not be found on a subsequent cache read. A *false miss* occurs, causing an unnecessary load from main memory but resulting in no data loss. It is also possible for the SEU to change a tag such that one or two CAM words incorrectly match a cache read request. This architecture requires the entire cache to be invalidated, but this can be done safely as the write-through architecture guarantees a valid copy in main memory.

Also note that this cache architecture does not include Forward Error Correction (FEC) to detect or correct SEUs in the data block. SEUs cannot be detected and soft error tolerance will be very poor.

The estimated number of devices in Cache B (the simple CAM) is shown in Table 8.4.

8.3.2.2 Fault Tolerance Analysis. This section analyzes the manufacturing yield of the simple CAM-based cache architecture.

There is very little fault tolerance capability in this version of the cache architecture. The CAM array possesses spare rows that can be used without restriction. Memory models for *Global Spares* are applicable for the CAM array. The remainder of the cache can be modelled using (5.3) and (5.11) for modules without fault tolerance.

The transmission gates connecting each CAM word to the output buses must be handled differently from the remainder of the CAM word. In this case, it is useful to extend fault analysis from the Boolean level to the device level. Two common failure modes of transistors are *fail-open* and *fail-closed* faults. With a fail-open fault, one of

Table 8.4: Cache B (Simple CAM) Device Count

Cache Size	Number of Devices per Mod		
	64KB	512KB	1MB
CAM Word	27,766	27,766	27,766
CAM Array (no spares)	28.4M	227.5M	454.9M
CAM Array (10% spares)	31.3M	250.2M	500.4M
CAM Word Decoder †	49,176	491,550	1.05M
Output Muxes	10,252	10,252	10,252
Output Registers	4,356	4,356	4,356
BIST/R	20,000 - 80,000		
Control Logic	20,000 - 80,000		
CAM Status Register†	40,572	324,432	648,828
NCW Register †	376	478	478
Control Module (total)	61K-121K	345K-385K	669K-709K
N_{cache} (10% spares)	$\approx 31.4M$	$\approx 228.3M$	$\approx 502.2M$

†: Assuming 10% spare rows.

the two transistors in the transmission gate cannot close, and thus a strong connection between the CAM word and the data bus cannot be made. The CAM word cannot be used to store data, but it is possible to disable the word and replace it with one of the spares in the array. A more serious problem is the fail-closed fault. With this fault, the register in the CAM word is *always* connected to the data bus line, resulting in bus contention when another CAM word tries to drive the bus. The result may be a short between power and ground and an indeterminate logic state on the line. A fail-closed fault on *any* of the transmission gates connecting to a data bus line will effectively disable the whole line even if it occurs in an unused CAM word.

Assuming the worst case *fail-closed* faults, modelling of cache yield requires the bus transmission gates be included in the “no fault tolerance” support logic section rather than the global spares CAM word analysis. For the Poisson case, the overall cache yield is thus

$$Y_{cacheB} = P(C) \cdot P(T) \cdot P(S), \quad (8.1)$$

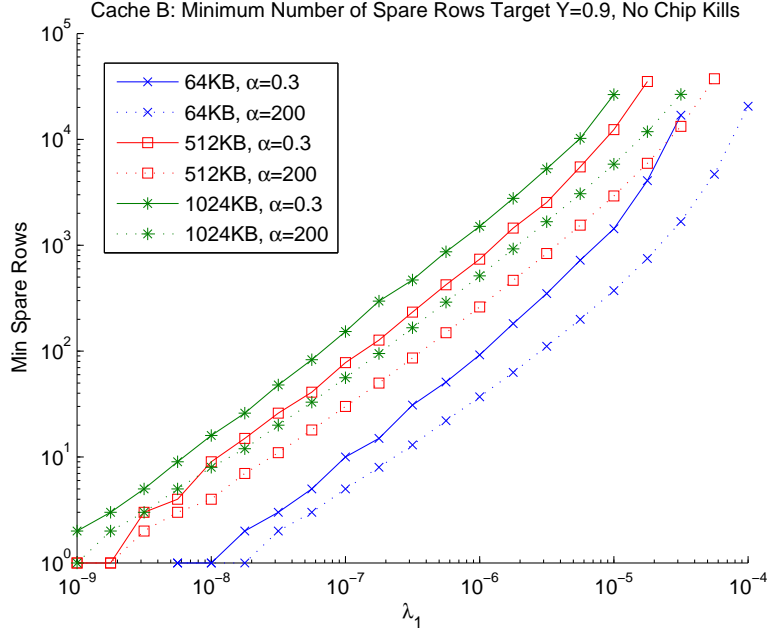


Figure 8.6: Minimum number of spare CAM rows to achieve a yield of 0.9 for CAM version B. Chip kill effects of the bus TGATES are not considered.

where $P(C)$ is the probability the CAM array is correctable (i.e., possesses at least R functional CAM words), $P(T)$ is the probability no bus transmission gates are faulty, and $P(S)$ is the probability that the remaining support logic is functional.

Ignoring the bus transmission gates for now, global sparing using spare CAM words is very effective for a MADP of roughly 10^{-5} (Figure 8.6). The target yield of 90% can be achieved using a relatively small number of spare rows. For $\lambda_1 > 10^{-5}$, the required number of spare rows grows exponentially and requires an unacceptable number of redundant devices. Note that this figure does not include chip kill effects of failing bus transmission gates. A failure in any bus TGATE will disable the entire bit column. Since no fault tolerance is included, this disables the entire CAM array. Adding spare rows increases the probability of a chip kill event and rapidly counteracts any benefit. When bus chip kill events are included, the plots in Figure 8.6 become much steeper and the slope increases to infinity. This affects overall cache yield. As shown in Figure 8.7, use of more than 10% spare rows provides diminishing benefit.

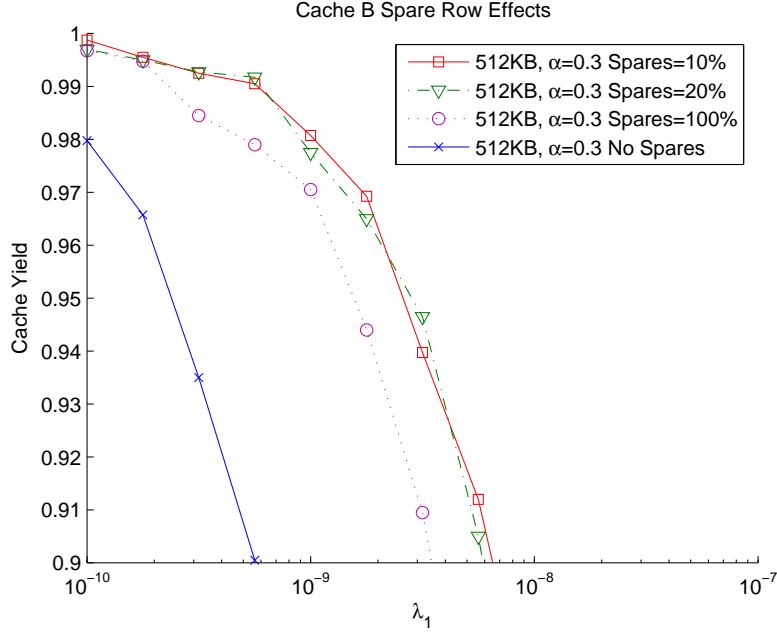


Figure 8.7: Due to the chip kill effects of the bus TGATES, adding spare rows soon becomes counterproductive.

For device technologies with $\lambda_1 < 10^{-5}$, this type of CAM array provides acceptable yields. Simulation results for the 1MB CAM array, not including support circuits, are shown in Figure 8.8. The two left lines show a CAM with no spare rows for the unclustered (Poisson) and the clustered cases. Yield is less than 90% even for defect rates of 10^{-9} . The other two lines show yield for the CAM with 8192 spare rows (i.e., 100% spares).

Most fault tolerant memory research concentrates on techniques which add spares to the main memory array. In this cache, however, the limiting factor is the support logic and the transmission gates rather than the CAM words.

The remaining devices in the cache (i.e., the support logic) possess a MADP of much less than the value of the CAM array. Figure 8.9 shows the yield of the support logic for both the Poisson case and that of defect clustering. Due to the large number of devices not protected by fault tolerance techniques, the MADP for the three cache sizes range from roughly 10^{-7} to 10^{-6} , much less than the defect tolerance capability of the CAM array with sufficient spare rows.

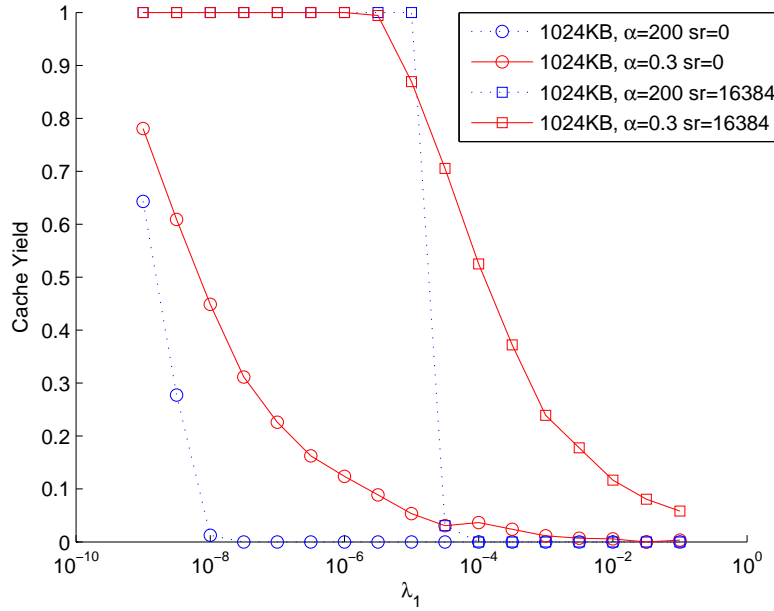


Figure 8.8: Examining only the CAM array without considering chip kills in the bus transmission gates, adding spare rows has a significant benefit to MADP.

An even more significant limitation comes from the transmission gates connecting the CAM array to the output bus. Even the small 64KB cache with no spares requires more than 2.1 million devices to make the connections. This total dominates the number of devices in the remaining no fault tolerance section of the cache. The yield of the transmission gate section of the cache is shown in Figure 8.10. The MADP of the three caches ranges from roughly 10^{-9} to 10^{-7} . These results are only marginally better than those of Cache A. Thus, without significant attention to the support logic and overall architecture, a fault tolerant memory core has little impact on overall cache yield.

8.3.2.3 Simulation. Cache B yield was simulated using MATLAB[®] to verify the accuracy of the analytical expressions for the Poisson models for the non-clustered case. The simulation initially calculates the number of devices in each module and in the entire cache. During each iteration of the simulation, a random number of faults are generated using the negative binomial distribution, with the

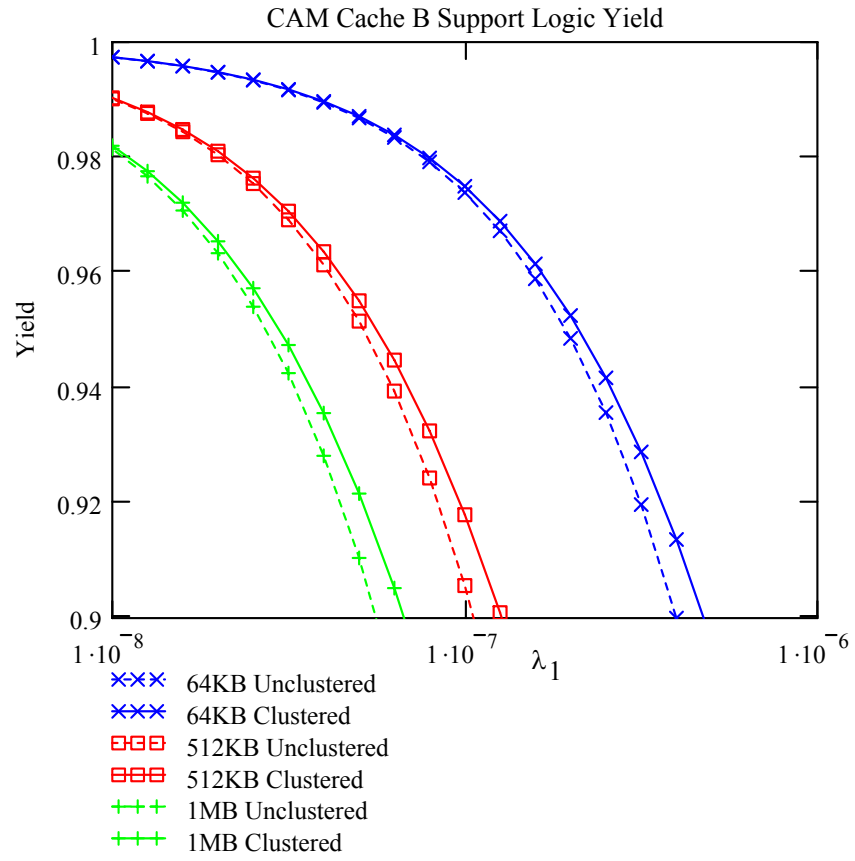


Figure 8.9: With no fault tolerance capabilities, the support logic is the limiting factor to MADP. Depending on the exact number of devices in the control and BIST modules, the MADP is between 6×10^{-7} and 7×10^{-5} .

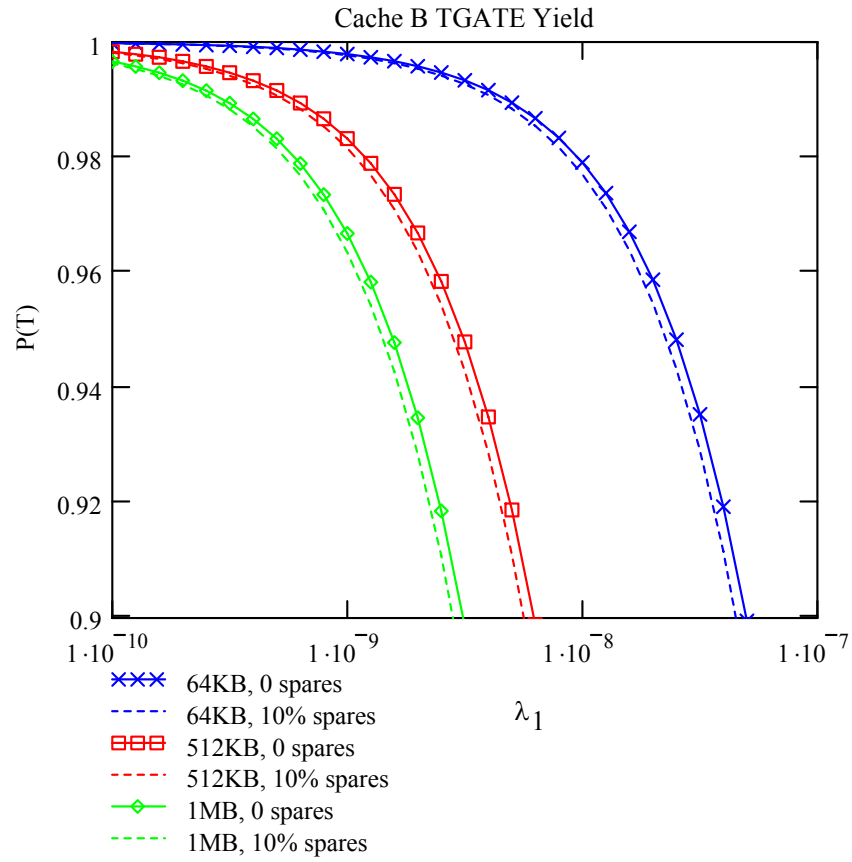


Figure 8.10: With no fault tolerance capabilities, the transmission gates connecting the CAM words to the bus are a limiting factor for MADP.

proper clustering parameter, α , and scaling parameter p based on the number of devices in the cache. For $\alpha > 10$, the negative binomial distribution approximates the Poisson. These faults are randomly distributed across the cache using a uniform distribution appropriate for large-scale defect clustering. Each module is examined for correctable faults. The entire cache is functional if all the modules are correctable, and at least rr of the tr CAM rows are correctable. At least 2000 iterations were run for each simulation. The average yield and 90% confidence intervals were computed and the confidence intervals confirm the average yield is within 1% of the true value.

The overall yield of cache B, combining the three sections, is shown in Figure 8.11. The MADP for a 90% yield is summarized in Table 8.8 at the end of the chapter. Comparing the results for Cache B to those of Cache A (incorporating no fault tolerance), Cache B provides only a small benefit to MADP for the medium and large caches. For the small 64KB cache, MADP is actually worse than the non-fault tolerant cache. The large number of devices required to implement the transmission gate connections and the CAM word select decoder effectively overcome the benefit of redundancy.

8.4 FDT Cache Memory Overview

This section proposes a new design for the CAM-based cache, called Cache C, incorporating improvements to correct the deficiencies of the basic CAM described in the previous section. The new design and operation of the improved cache is described, and its yield analyzed. This section provides a summary of the design, including schematics and hardware cost figures. Operation and test of the cache are discussed in Section 8.5. The design is discussed in detail in Section 8.6, including detailed analysis of ECC code choices, bus design, and support logic fault tolerance. Yield simulation results are given in Section 8.8.

The CAM-based cache described in the previous section suffers from three key drawbacks: the design uses a large number of devices in areas unprotected by fault tolerance schemes; the bus design allows a single fail-closed defect in a transmission

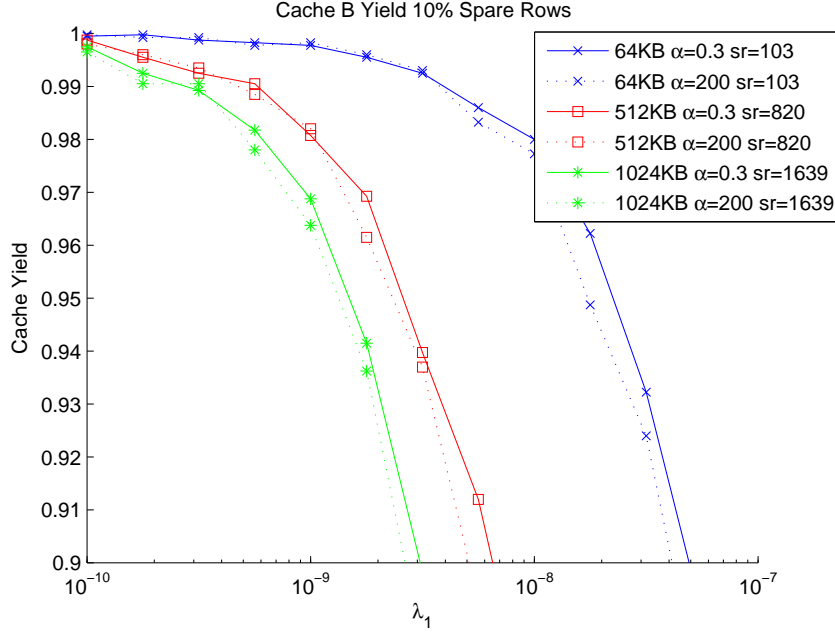


Figure 8.11: The combined yield of the basic CAM cache, utilizing 10% spare row coverage.

gate to disable an entire line in the output bus; and there is no tolerance for soft errors or single-event upsets (SEUs) in the control module or the tag and data memory in the CAM array. Furthermore, there is no way to determine the sources of conflicting hits, and no way to recover without invalidating the entire cache.

The architecture in this section corrects these deficiencies. The overall approach is to make best use of the CAM array's ability to replace failed components with spare rows by moving logic from the support and control sections into the CAM array, where it is easily protected using spare rows. Highlights of the CAM word design include: the use of the extended Golay (24,12) code for SEU protection; replication of the output bus signals to protect against failures in the transmission gates; internally-enabled transmission gate multiplexing to reduce the number of output bus signals required; and the use of an internal CAM Word address comparator rather than a monolithic external address decoder.

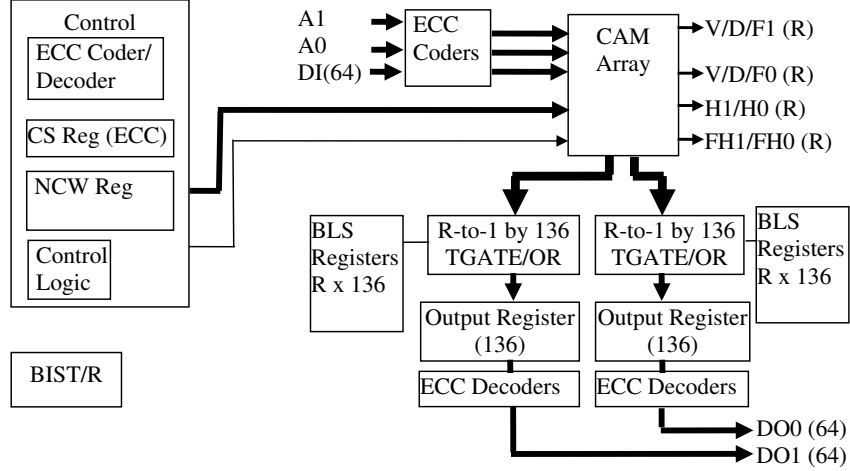


Figure 8.12: The improved CAM architecture incorporates ECC in the data memory and control registers, distributes the function of the CAM word decoder among the rows in the CAM array, and replicates each bus line R times to reduce the probability an output line will be disabled.

8.4.1 CAM Top Level Design. The top level architecture of Cache C is shown in Figure 8.12. In addition to ECC in the CAM array, it adds ECC protection for the registers used in the control and BIST/R modules. The ECC coders and decoders for the data memory are external to the CAM array and are protected by triple modular redundancy (TMR). Finally, the R replicas of each signal are combined into a single logic signal using OR gates, with inputs protected by transmission gates controlled by the new Bus Line Status (BLS) registers. In this way, if a bus TGATE suffers a fail-closed fault, that column can be disconnected from the OR gate using the TGATE at the gate input. Thus, as long as at least one of the R columns has no faults, the signal can be corrected.

The implementation is fully-associative, and thus a cache word of 64 bytes can be placed in any of the CAM entries. Depending on the cache size, the CAM requires 1024, 8192, or 16384 entries, not including additional entries used as spares. The cache has two read ports. To perform a read operation, the read address is placed on either the $A1$ or $A0$ address bus. Each of the CAM words compares the addresses on this bus with the address of their stored data. If the tag addresses match, internal

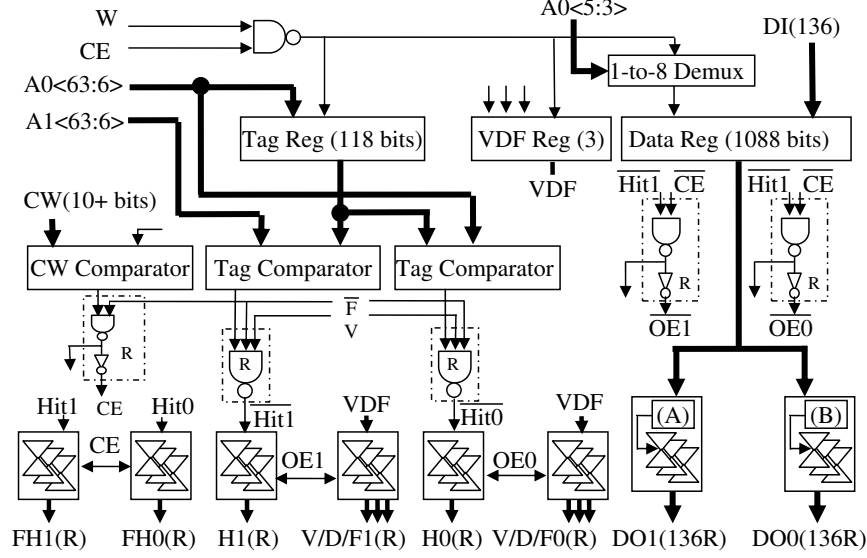


Figure 8.13: The improved CAM word incorporates ECC in the data and tag memory, replaces the single CAM-level address decoder with distributed CAM address comparators, and replicates each bus line R times. The extended Golay (24,12) code is used to encode each of the eight 64 bit words in a CAM row into six 24 bit codewords. The upper 8 bits of the sixth codeword are zero, and are not stored. The same approach is used for the tag bits.

multiplexors inside each CAM word select the registers of the desired encoded word to connect to the *Data Out (DO)* bus.

Normal operation of the fault tolerant CAM cache is similar to Cache B, with only a slightly longer latency due to the fast parallel ECC encoders/decoders.

8.4.2 CAM Word Design. The improved CAM word design is shown in Figures 8.13 and 8.14. The CAM word consists of registers to store the tag and data fields, as well as word status (e.g., valid and dirty bits, plus an additional bit to store the fault status of the CAM word).

The (24,12) extended Golay code protects against hardware faults and SEUs in the tag and data registers. Each of the eight 64 bit words in a CAM row are encoded into five 24 bit codewords and one 16 bit codeword. The upper 8 bits of the sixth codeword are zero and are not stored. Thus, the eight 64 bit data words in a cache

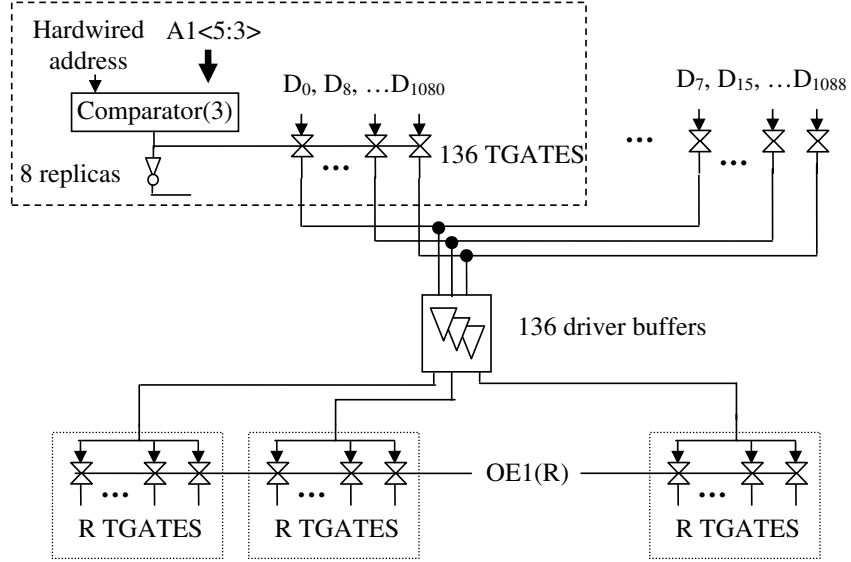


Figure 8.14: To select the proper 136 bit codeword from the 1088 bits stored in the CAM word data section, distributed comparators are used.

block require 1088 bits of storage. Likewise, 118 coded bits store the 58 bit uncoded tag.

A 1-to-8 demultiplexor is used for cache writes, as the memory architecture uses 64-bit words. Thus, a single cache write changes only 136 bits in the 1088 bit CAM word. The demultiplexor enables the Write Enable lines for the proper 136 register flip-flops.

Transmission gates connect the CAM word to the output bus. To limit the number of bus lines, the comparator/transmission gate (TGATE) modules in Figure 8.14 act as a multiplexor to select a single 136 bit codeword to place on the data bus. Eight three-bit comparators hardwired to the proper offset in the cache line enable the desired group of 136 TGATES. Driver buffer gates are included after the TGATES to restore signal levels and increase speed. Each bus line is replicated R times to protect against faults in the bus connection TGATES. CE and OE enable logic is replicated to prevent faults in these gates from connecting the row to all R lines of a bus line simultaneously.

Table 8.5: Cache C Device Count

Cache Size	Number of Devices per Mod		
	64KB	512KB	1MB
CAM Word ‡†	58,898	63,704	63,722
CAM Array (10% spares)	66.38M	574M	1.148B
ECC Encoders (data/tag)	62,304		
ECC Decoders (data)	91,440		
Control BLS Regs/ORs/Regs	3,060	6,100	6,100
Data BLS Regs/ORs/Regs	93,024	175,712	175,712
BIST/R	253,756		
Control Logic	253,756		
CAM Status Register †	81,216	649K	1.30M
NCW Reg/ECC coders †	13,660	13,762	13,796
Control Module (total)	348.6K	916K	1.57M
Chip Kill Devices †	15,880	15,982	16,016
$N_{cache}†$	67.23M	576M	1.151B

†: Assuming 10% spare rows

‡: Including bus transmission gates

The control module uses the CAM Word (CW) comparator to select a single CAM word. This is useful for testing and in the event of a conflicting hit. For a conflicting hit, the control module can determine which rows match the tag and invalidate only those rows. Without this capability, the entire cache must be invalidated and reloaded.

8.4.3 Hardware Cost. Hardware costs for the FDT cache design were calculated using the techniques developed in Chapter V. The estimated number of devices in Cache C is shown in Table 8.5. The device total is dominated by the number of devices in the CAM array. As with Cache B, 10% sparing is used.

8.5 Operation and Testing

8.5.1 Normal Operation. Normal operation of Cache C is very similar to that of Cache B with the exception of the additional propagation delay to encode and decode the data and tag words. Using ECC should not have a large impact on cache latency, as there are fast parallel implementations of both the encoders and

decoders for extended Golay codes [BMH00,LDJC83]. One parallel design for the extended Golay decoder uses 617 logical gates (roughly 2468 devices) [BMH00]. A parallel encoder is easily designed using the algorithm in [LDJC83] and requires 1744 devices.

The CAM Word decoder used during cache writes is replaced by an address comparator in each CAM word. In this way, a failure in the decoder disables a single CAM row rather than the entire CAM. The output multiplexers are moved into the CAM word for the same reason. As shown in Figure 8.14, rather than using a multiplexer to select the desired 64 bit (uncoded) word from the 512 bit cache block (uncoded), eight registers are connected to a single output bus line by transmission gates. Eight three-bit comparators hardwired to the proper offset in the cache line enable the desired group of 144 TGATES. Driver buffer gates are included after the TGATES to restore signal levels and increase speed.

8.5.2 Cache-Specific Error Behaviors. A cache can be either write-back or write-through. In environments subject to SEUs, a write-through architecture is preferred. If a SEU changes a bit in a tag address stored in the cache, a requested cache block may not be found on a subsequent cache read. A *false miss* occurs, causing an unnecessary load from main memory, but results in no data loss. A *false hit* occurs when an SEU in a tag bit causes a CAM word to incorrectly match a cache read request. If two or more CAM words match the request a *conflicting hit* occurs.

8.5.3 Testing and Recovery. Testing of the cache is done in three phases. The first phase tests the support modules (i.e., control, BIST/R, encoders, decoders, etc.) for faults. The second phase tests the control and data busses. For the control signals, at least one of the R replicas of each signal must be functional or the cache is uncorrectable. For the data signals, the error correcting code overcomes faults in the output registers, the TGATE/OR modules, BLS registers, R signal replicas, and the CAM array data registers. The BIST/R module tests which output signals can be

corrected using the bus replica signals and ECC code. Finally, the third phase tests the CAM array.

Testing of a CAM row includes all of the logic shown in Figure 8.13 including the output bus transmission gates. The extended Golay code corrects up to 3 faulty bits in every 24 bit code word in the data registers. The BIST/R reserves one bit of correction capability to correct Single Event Upsets (SEUs) during operation. The remaining two bits of error correction can be used to overcome manufacturing faults. Of the tr total CAM rows, at least rr must be functional or the cache is declared uncorrectable.

False hits, conflicting hits, and false misses are handled as follows. Tag words are encoded using the (24,12) extended Golay code. All read or write tags are encoded before being sent to the CAM Array. Decoding is not required, as comparisons are done on the coded words. Handling of false misses is identical to the previous CAM cache. A *write-through* cache design should be used to ensure a valid copy of the data is in main memory. If a tag word suffers a SEU, a cache read for that block generates a *false miss* and the block is reloaded from main memory. The erroneous cache block is overwritten by subsequent load operations.

Conflicting and false hits are handled more effectively than the previous CAM cache. Probabilities of *false* and *conflicting hits* are minimized by encoding the tag bits using an ECC. With a Hamming distance of eight, at least eight bits in a 24 bit codeword must be in error to generate another valid codeword. Additional protection is provided for *conflicting hits*. When the conflicting hit occurs, two or more CAM rows generate hit signals and attempt to drive the data bus. The control module detects the invalid voltage levels during bus contention, suspends operation, and locates the specific CAM words causing the hits. The CAM Word (CW) address selects each CAM row individually. Thus, only the CAM words matching the tag generating the conflicting hit are invalidated, minimizing downtime and eliminating the need to invalidate the entire cache. The BIST/R module tests the suspect CAM rows to de-

termine if the cause was temporary (i.e., SEU or soft error), or is a permanent failure. In the latter case, the CAM row is marked nonfunctional and removed from use.

The testing approach is aggressive, and requires the BIST/R to examine the output busses and the CAM words together. To reduce test complexity, an alternative approach tests the CAM rows separately from the output busses, and uses the ECC capability for either the data registers or for the output busses and registers, but not both. This requires either more spare rows in the CAM array or the use of additional fault tolerance logic in the output registers to avoid adding roughly 9800 devices to the chip-kill portion of the design. If test times allow, however, maximal use of the ECC capability is preferable.

For each CAM word, the BIST/R tests the registers, comparators, and output TGATES for correct operation. Fault status is stored both in a register in each CAM word as well as the CSR in the control module. Both the BIST/R and the control module use TMR-protected module reconfiguration.

Status of the CAM words is stored in the CAM Status Register (CSR) in the control module. As shown in Figure 8.12, the control module of the cache consists of four parts: a *CAM Status Register (CSR)*, the *Next CAM Word Register (NCWR)*, an ECC encoder/decoder, and control logic. Upon startup, and periodically during operation, the Built-In Self-Test/Repair (BIST/R) module tests each CAM word. Faulty words are replaced with spare words in the CAM array. The NCWR determines which CAM word to use for the next cache write operation. The cache uses a modified least-recently used strategy for block replacement. The control logic block includes the remaining combinational and sequential logic to manage cache operation. The *tr* bits are encoded using the (24,12) extended Golay code to protect against faults and SEUs in the CSR. The Next CAM Word Register (NCWR) is so small that ECC is not required.

Fault status of the output bus signals is stored in the Bus Line Status (BLS) registers. As shown in Figure 8.15, the registers disconnect faulty bus signals from the

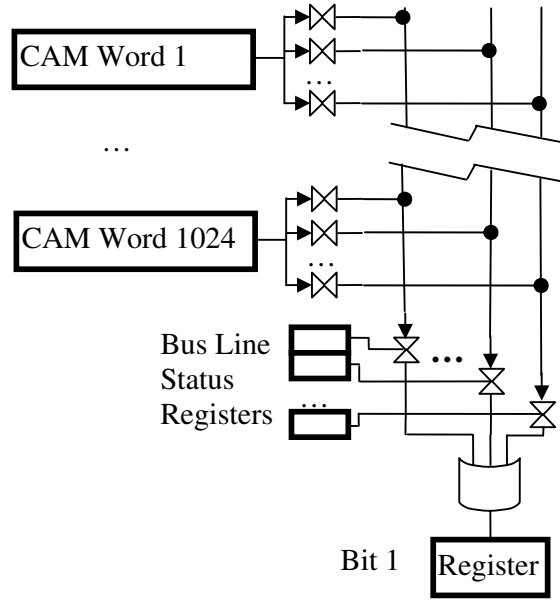


Figure 8.15: Redundant bus architecture.

OR gates combining the functional signals. This approach requires fewer gates than an R -to-1 multiplexer, as well as providing additional protection against transmission gate faults. For example, CAM row 1 output bit X can connect to column A but not to column B. CAM row 2 output bit X can connect to column B but not to column A. With a multiplexer, only one of these two columns can be used as output bit X . Rather than requiring all rr utilized rows to connect to the same column, the OR gate means both columns A and B can be used.

The ECC encoders and decoders for the tag and data memory are external to the CAM array and protected by TMR. Finally, the R replicas of each signal in the data and control bus are combined into a single logic signal using OR gates, with inputs protected by transmission gates controlled by the new Bus Line Status (BLS) registers (Figure 8.15). In this way, if a bus TGATE suffers a fail-on fault (i.e., a fault that causes the device to be permanently *on*), that column can be disconnected from the OR gate using the TGATE at the gate input. Thus, as long as at least one of the R columns has no faults, the signal can be corrected.

8.6 Design Choices

This section describes the proposed design in detail. Several design alternatives and their impacts on module and overall cache yield are examined. The design alternatives include: choice of error correcting code, replicated busses, logic module fault tolerance, and row sparing requirements for the CAM array.

Overall yield is the product of the independent modules (5.11). The modules can all be analyzed separately, with the exception of the bus logic and the CAM array for reasons that will be explained in a later section.

8.6.1 Error Correcting Codes. The probability a single codeword can be corrected is from (5.24),

$$Y_{ECC} = \sum_{k=0}^t \binom{C}{k} (1 - Y_{bit})^k (Y_{bit})^{C-k}, \quad (8.2)$$

where c is the number of bits in the codeword (i.e., 24), Y_{bit} is the probability a single bit register will be functional, and t is the error correcting capability of the code (in bits).

Seven coding schemes are examined to determine the most effective technique. TMR simply replicates each of the data bits three times, and can correct one error per three bit codeword. A (7,4) Hamming code with $t = 1$ encodes each 64 bit word into 16 seven bit codewords. The next four alternatives are variations on the (24,12) extended Golay code. Since the 64 bit output word is not evenly divisible by 12, four bits remain after the first 60 bits are encoded in 5 codewords. The remaining four bits can be either: left uncoded, encoded with the (7,4) Hamming code, tripled and encoded with extended Golay, or zero-padded and encoded with the (24,12) extended Golay code. In the latter case, eight of the 24 coded bits are always zero and do not have to be stored in the registers. Finally, the (72,64) code used in the Athlon 64 data cache is also considered.

The unclustered yield for a 64 bit uncoded cache word was derived analytically and is shown in Figure 8.16. *Maximum Allowed Defect Probability* (MADP) is the upper limit on mean device defect probability, $\bar{\lambda}_1$, such that a specified module yield can be achieved (e.g., 90%). The extended Golay code options have higher MADP values than TMR or the simple (7,4) Hamming code. The best results are obtained by zero-padding the upper four bits of the 64 bit word and encoding using extended Golay. The result is very similar to the approach in which the four bits are triplicated and then encoded (i.e., making use of all 12 of the bits in the uncoded word). However, the triplicated code still only corrects three errors. For the zero-padded approach, the three bit error correction capability is applied to the 16 coded bits (since 8 are always zero and can be omitted) rather than to a longer 24 bit codeword.

Extending these results to an entire 512 bit CAM row, the predicted yield for a CAM row is shown in Figure 8.17. Again, the latter extended Golay approach provides the best defect tolerance and is adopted for the design. For the extended Golay code, $t = 3$. In this cache architecture, the error correction capability is split between hard faults and SEUs in the registers. The BIST/R module reserves one bit per codeword for SEU correction. Thus, for the remainder of the cache yield analysis, $t = 3 - 1 = 2$.

8.6.2 Bus Design.

8.6.2.1 Description. The output signals in each CAM word are connected to the common bus using TGATES controlled by the output enable signal. Without fault tolerance, a single fail-on fault in any of the transmission gates can result in more than one CAM word being connected to a column simultaneously. Bus contention results in an indeterminate logic state on the output, effectively disabling the entire column. The probability of a column disabling fault increases as the total number of rows increases.

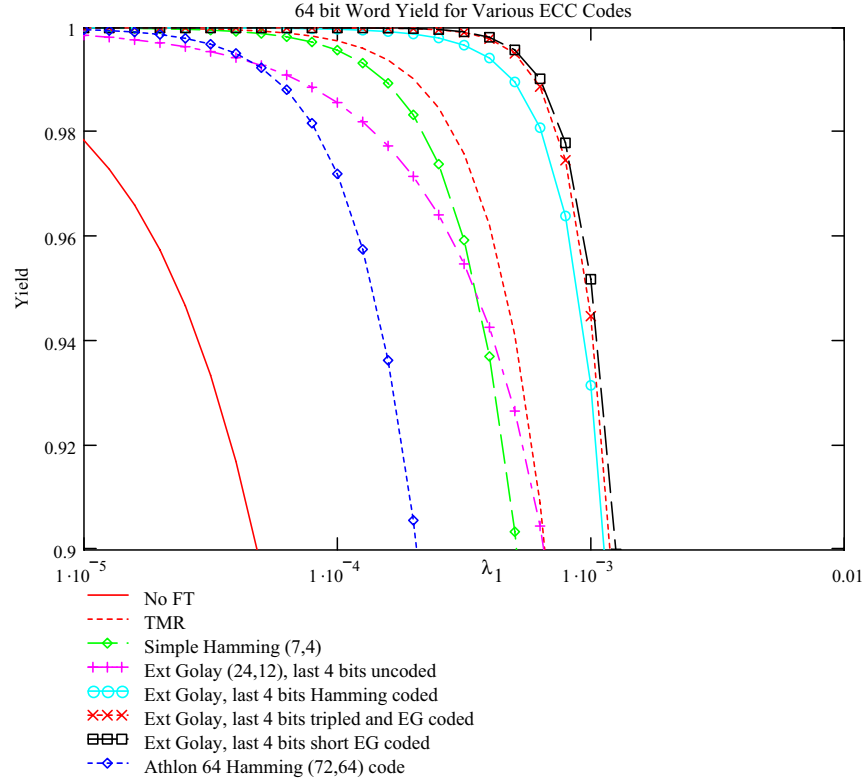


Figure 8.16: Predicted yield of 64 bit word with different ECC strategies. The (24,12) Extended Golay and (7,4) Hamming code are considered. A cache block is eight 64-bit words. Thus, a non-integer number of extended Golay codewords are required to encode the 64 bit word. Encoding of the last four bits of the word can be handled in different ways, with different results.

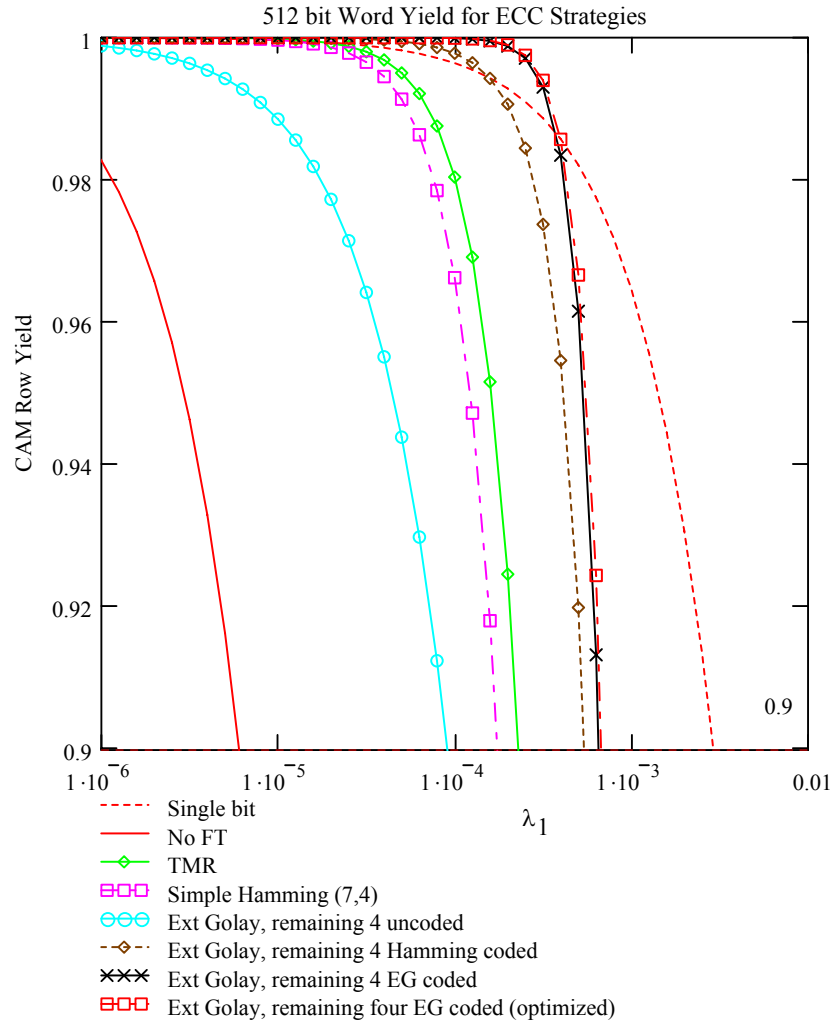


Figure 8.17: Predicted yield for 512 data bit CAM word using various ECC approaches. Extended Golay coding of each 64 bit word (using 6 Golay codewords) provides the greatest MADP.

A fault tolerant design based on R -fold column replication is shown in Figure 8.15. The R columns are combined using an OR gate. Faulty columns can be disconnected from the OR gate using the bottom TGATES, whose states are controlled by the Bus Line Status Registers. This approach requires fewer gates than a R -to-1 multiplexer, as well as providing additional protection against transmission gate faults. For example, suppose CAM row 1 output bit X can connect to column A but not to column B. CAM row 2 output bit X can connect to column B but not to column A. With a multiplexer, only one of these two columns can be used as output bit X . Rather than requiring all rr utilized rows to connect to the same column, the OR gate allows the use of both columns A and B.

One BIST/R improvement is made by considering faults at the device level rather than the Boolean level. An example is shown in Figure 8.18. If fault modelling includes both fail-closed and fail-open faults rather than simple failures, fail-open faults can be tolerated in the transmission gates of CAM rows used to store data as long as at least one of the R replicas of that output is both connectable to its signal bus, and the bus is functional (i.e., has no other fail-closed faults in other rows). This increases the probability an output signal bus is correctable, requiring fewer spare CAM rows.

In addition to column replication, the data bus is also protected by ECC. Since the decoders are placed outside the CAM array, ECC corrects faulty column busses in addition to the CAM word data registers.

8.6.2.2 Bus Yield Models. Replication of the control and data output busses prevents a single fail-closed defect in one of the transmission gates connecting a CAM row to the bus from disabling the entire column. Analytical expressions for the yield of the control bus are straightforward, and are shown in this section. The expressions for the data bus are similar, but must also incorporate the effects of error correction provided by the extended Golay code.

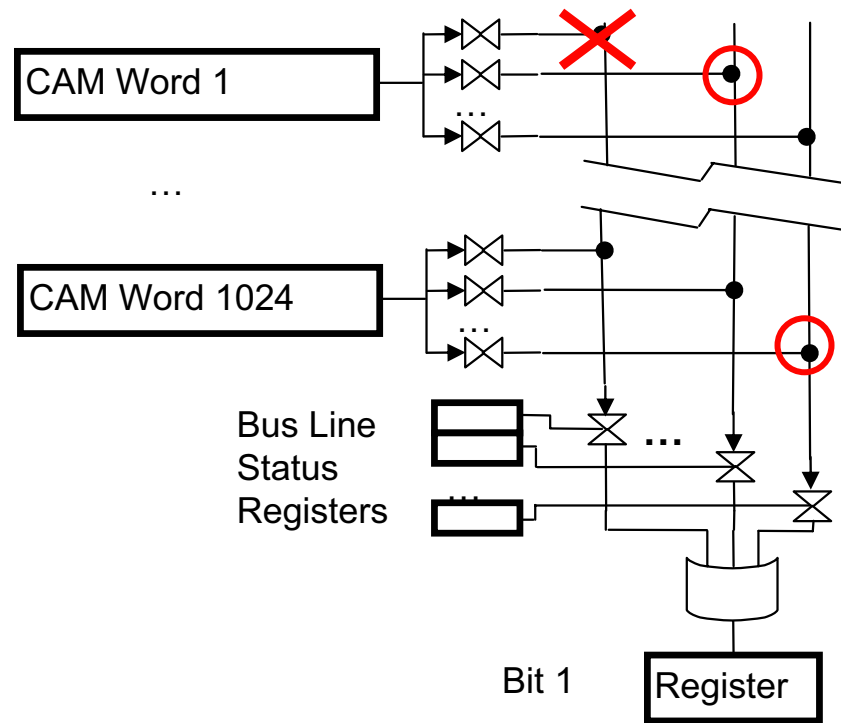


Figure 8.18: If fault modelling is done at device level, reliability can be improved. Here, the 'X' indicates a fail-closed fault and 'O' indicates a fail-open fault. At the Boolean gate level, none of the three columns are fault-free, and thus the bus is unusable. By modelling faults at the device level, column 2 can be used for CAM Word 1024, and column 3 can be used for CAM Word 1. The bus is correctable.

Each bus signal is made up of three parts: module A contains the output register, the Bus Line Select (BLS) status register, the OR gate for each signal, and the TGATES connecting each replicated column to the OR gate. All of these devices must function for the signal to be correctable. All of these devices must be functional as well, since a fail-closed fault can connect a signal column permanently to the OR gate. Module B contains the TGATES connecting the CAM words to the busses. Failures in these TGATES can be tolerated, so long as at least one B module contains no faults. The probability a bus signal is correctable is

$$P_{bus} = P_{modAfunc} \cdot P_{modBcorr}. \quad (8.3)$$

For module A, the yield expression is derived from (5.3). Module B represents the R replicated columns of tr TGATES. At least one of these columns must contain no faults for the bus to be correctable. That is,

$$P_{modBcorr} = 1 - P_{allRcolsfail} = 1 - P_{onecolfails}^R. \quad (8.4)$$

.

The probability a single column fails is

$$P_{onecolfails} = 1 - P_{onecolfuncs}. \quad (8.5)$$

The probability a column functions is found using (5.3) with the number of devices derived from the number of TGATES in the tr rows, plus the NAND, NOR, and NOT gates controlling the enable lines inside the CAM word. Faults in these gates may cause the transmission gate to connect to the column.

The fault tolerance of a single output protected using the bus replication scheme is shown in Figure 8.19. The plot shows only the unclustered (Poisson) defect model, but looks very similar for the clustered case. For caches with 100% sparing (i.e.,

Table 8.6: Maximum Allowed Defect Probability for 90% Yield for Various Bus Options (unclustered).

Cache Size*	64KB	512KB	1MB
Control (no FT)	5.8×10^{-7}	7.3×10^{-8}	3.7×10^{-8}
Control (CR)†	2.8×10^{-5}	8.1×10^{-6}	4.3×10^{-6}
Data (no FT)	1.9×10^{-8}	2.4×10^{-9}	1.2×10^{-9}
Data (CR)	1.1×10^{-6}	6.0×10^{-7}	6.0×10^{-7}
Data (ECC)	1.8×10^{-6}	2.3×10^{-7}	1.1×10^{-7}
Data (CR/ECC)†	4.7×10^{-5}	1.0×10^{-5}	5.1×10^{-6}

* CAM spare rows = 10% of total.

† CR: Col Replication. $R = 8$ for 64KB, 16 for 512KB/1MB.

$tr = 2 \cdot rr$), the maximum allowable defect probability without bus replication is between 10^{-8} and 3×10^{-7} . Using bus replication, yield increases greatly for even small numbers of redundant columns. For a factor of $R = 8$, MADP ranges from 10^{-4} to 10^{-3} for the three cache sizes. Larger levels of redundancy are not required for these caches, as other modules in the architecture become the limiting factor rather than the busses.

The predicted yields of five bus options are shown in Table 8.6. The results shown are computed analytically for the Poisson case and assume that the ECC decoders are functional. For the control bus, column replication increases MADP by two orders of magnitude for the larger caches. For the data bus, ECC by itself provides some benefit. Using both ECC and column replication increases the MADP for the data bus even further, by three orders of magnitude for the larger caches.

Column replication provides significant yield improvement for these modules at the cost of only a four to sixteen-fold increase in the number of bus TGATES. For the 1MB cache with 10% spare rows, approximately 40.7M devices are required for the bus TGATES. As shown in Table 8.7 at the end of the chapter, the bus represents only a small fraction of the overall device count.

8.6.3 ECC effect on CAM Array and Output Bus Yield. ECC is used to correct faults occurring primarily in the data registers in the CAM words, but can also

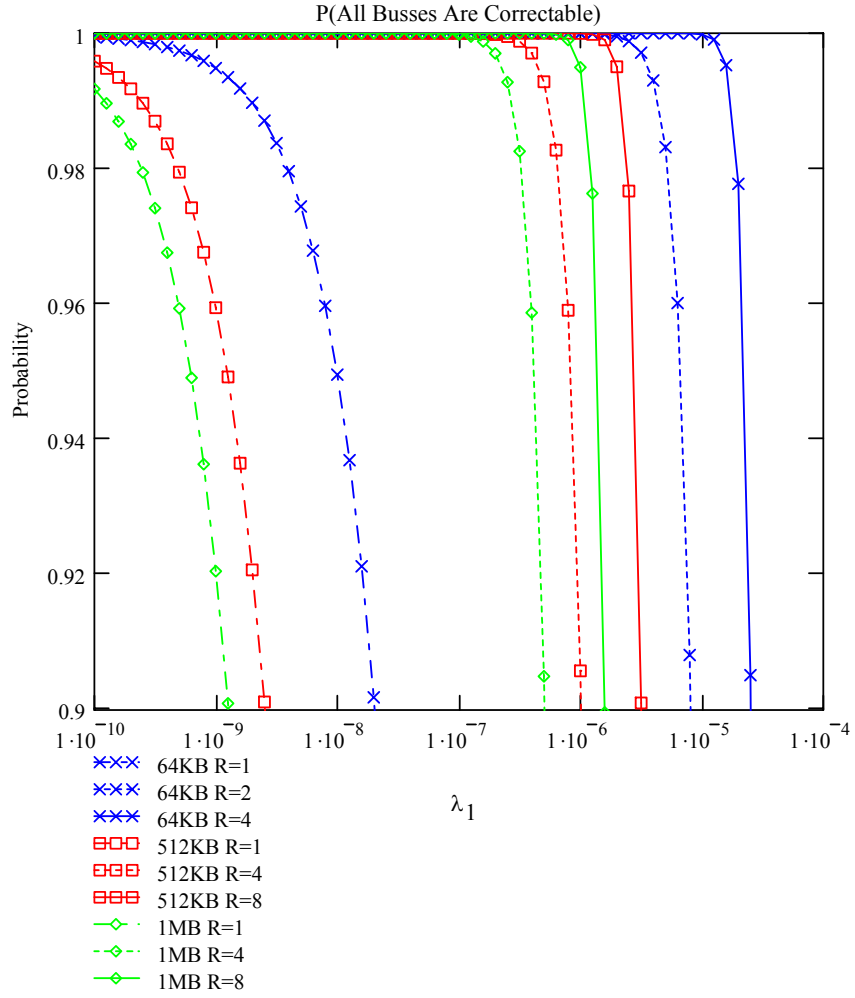


Figure 8.19: The probability the output busses are correctable for Cache C with various levels of redundant busses. This worst case assumes 100% spare rows for each cache size, and no spare rows to replace faulty rows. Bus sparing is very effective, resulting in MADP values in excess of 10^{-5} for only 4 or 8 copies of each signal.

correct errors induced by the output switching TGATES in the CAM word, failed bus lines in the data bus, and faults in the output registers. One approach for performance modelling assumes ECC is only used internally for each CAM word (and not used for the output bus or output registers), or vice versa. A more accurate model analyzes the dependent probabilities of the registers internal to each CAM rows, the output bus TGATES, and the output registers. Since the complexity of the architecture makes this impractical, the most effective way to estimate the yield of the CAM array and redundant bus scheme together with ECC is through simulation.

8.6.4 BIST/R and Control Modules. The number of devices in the control and BIST/R modules are estimates. Analysis of the yield for the individual components showed the generic control and BIST/R logic are the limiting cases when N_{BISTR} and $N_{control}$ are large. For simulation, each of the two modules are assumed to be a single module of 50,000 devices. TMR-protected reconfiguration is used for each module. It is also assumed each module has 128 outputs. If the actual number of devices in the modules is smaller, or the modules can each be broken into two or more independent submodules, yield would slightly improve. However, the ECC decoders soon become the limiting factor, and even if the control and BIST/R modules are ignored, overall yield does not improve significantly.

Three alternative fault tolerance approaches for the BIST/R module are shown in Figure 8.20. Analysis of the control module and other logic modules is similar. The number of devices in the BIST/R module is assumed to be $N_{BISTR} = 50,000$. The number of outputs is assumed to be $W_{BISTR} = 128$. From the figure, module-based reconfiguration offers the best performance, but no protection against soft errors. TMR-protected reconfiguration performs almost as well as reconfiguration, but provides additional protection against soft errors. For this reason, TMR-protected reconfiguration (TMR-R) was chosen to protect the BIST/R module.

Analysis of the BIST/R module makes two assumptions: dependence between the devices in the module, and overall module size, N_{BISTR} . If the logic is not

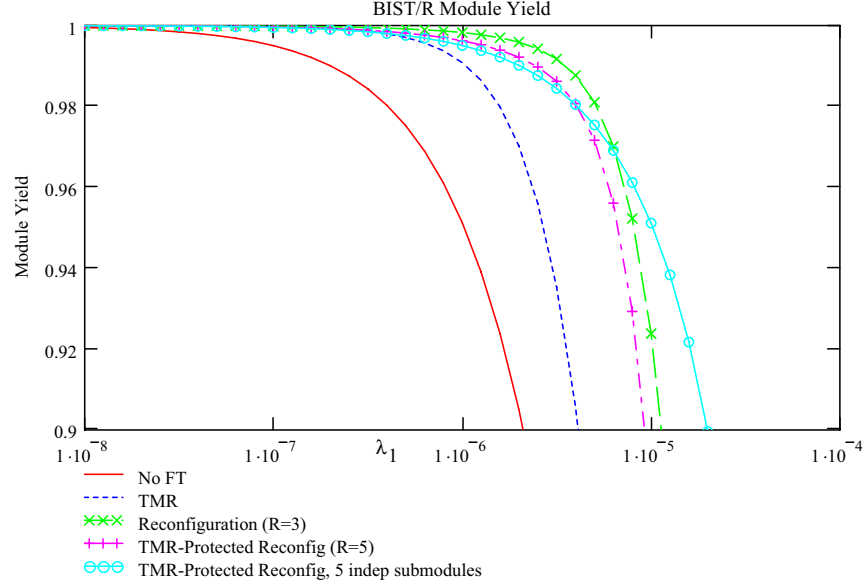


Figure 8.20: Module yield of the BIST/R. The use of TMR-protected Reconfiguration provides yield almost equivalent to module-based reconfiguration, plus additional protection against soft errors.

dependent, the BIST/R module could be broken into two or more submodules, each protected by its own fault tolerance scheme. The final line in Figure 8.20 shows the effect of breaking the BIST/R module into five independent submodules, each using TMR-protected reconfiguration. By assuming dependence, we perform a worst case analysis.

The second assumption is the number of devices. Figure 8.21 shows the unclustered model yield of the BIST/R module for four different defect rates (λ_1), versus module size, N_{BISTR} . For $\lambda_1 = 10^{-6}$, $Y \approx 1$ for $N_{BISTR} \leq 2 \times 10^{-5}$. Thus, accuracy in the module size estimate is not critical unless $N_{BISTR} > 2 \times 10^{-5}$. For $\lambda_1 > 10^{-6}$, yield rolls off with much smaller values of N_{BISTR} .

8.6.5 Other Modules. Similar analysis is performed on each of the remaining modules in the cache. TMR protects the ECC encoders and decoders rather than TMR-protected reconfiguration due to the relatively small number of devices in the encoders (1744) and decoders (2468). In these cases, MADP is only slightly better for

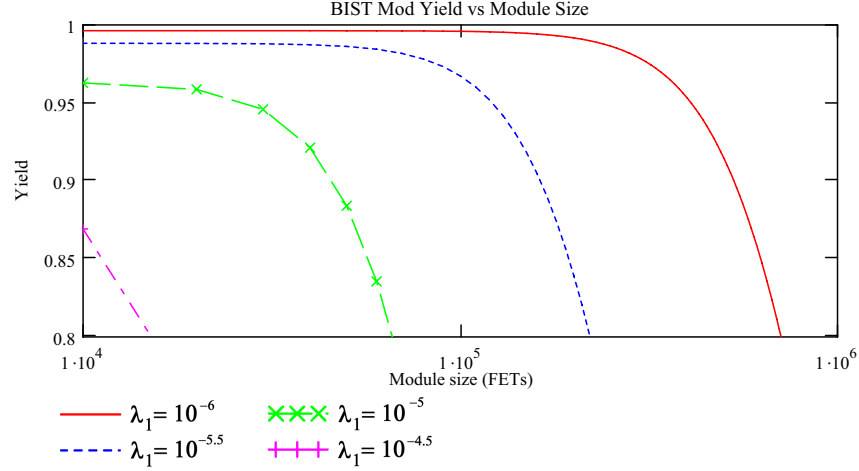


Figure 8.21: BIST/R Module yield versus the estimated module size.

TMR-R than for TMR, at the cost of almost twice as much hardware added to the chip kill section of the circuit. For example, for the single extended Golay encoder used in the control module, $MADP_{TMR} = 9.464 \times 10^{-5}$ and $MADP_{TMRR} = 1.190 \times 10^{-4}$, while $N_{ck_TMR} = 432$ and $N_{ck_TMRR} = 844$. Thus, the benefit to the module MADP is deemed insufficient to justify the additional devices in N_{ck} .

Extended Golay code is used to protect the CSR due to its large size (one bit per CAM row). Due to the small number of bits stored, the NCWR is left unprotected. rather than adding the additional switching needed to protect it with ECC.

8.7 Yield Simulation

The fault tolerance of the FDT cache design is tested through Monte Carlo simulation in Matlab. This section describes the results of the experiments to determine average yield, as well as the required number of CAM rows.

8.7.1 Simulation Methodology. The simulation first estimates the number of devices in each module. During each iteration, the simulator randomly generates the number of defects in the cache using the negative binomial distribution, based on the total number of devices in the cache, and with the clustering parameter α . Assuming

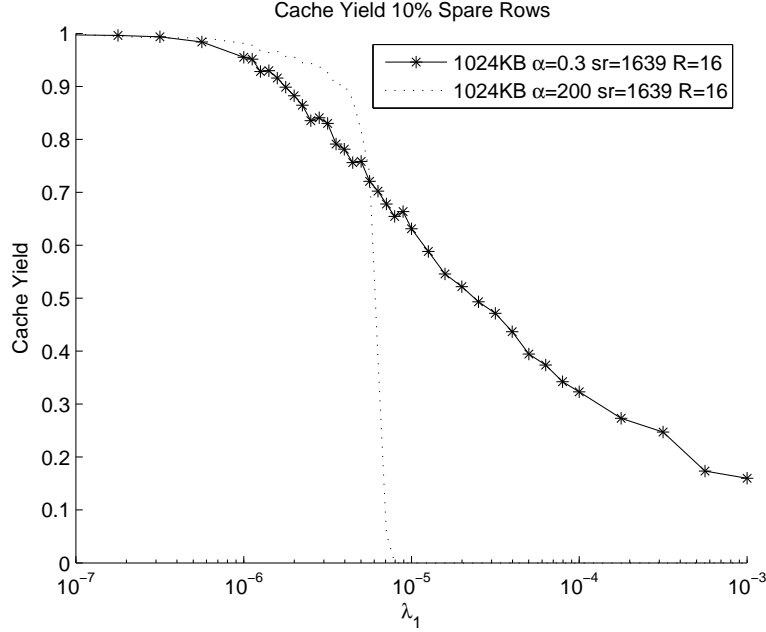


Figure 8.22: Simulated yield of the 1024 CAM cache for both clustered and Poisson defect models.

large-scale clustering, these defects are distributed uniformly among the devices in the cache. The simulator examines each module to determine whether defects can be corrected. Average yield is computed after running the simulation for at least 1000 iterations. The 90% confidence interval of the yield was computed to ensure error was no more than 1.5%.

8.7.2 Cache C Results. The simulated yield of the 1MB cache is shown in Figure 8.22. The CAM array has $CAM_{rr} = 16384$, with 1639 spare rows (10%). Both the unclustered (Poisson) and clustered yields are shown. While the yields differ greatly as the defect probability, λ_1 , increases, the values are very close in the range of greatest interest, where $Y > 0.9$. Thus, Poisson yield is a good approximation when designing large circuits for production.

The cache yield is the product of the yields of the individual modules. Simulating the CAM array by itself determines how rows are required for $Y_{CAM} \geq 0.9$. As shown in Figure 8.23, adding spare rows increases the MADP of the CAM array and

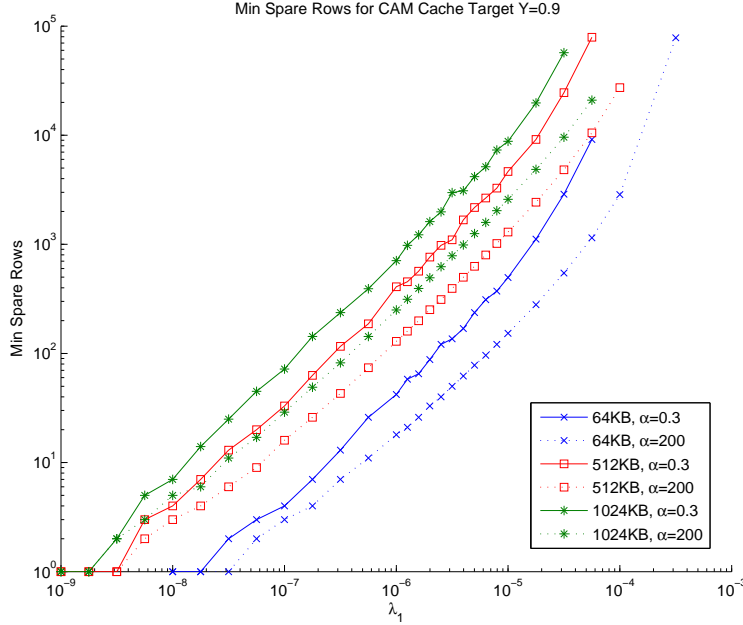


Figure 8.23: Minimum number of spare rows for the CAM array. Chip kill effects due to failures in the output enable logic in the CAM words are not shown, and increase the slope greatly. This forms an upper bound on the useful number of spare rows.

increases overall yield. Only a moderate number of spare rows (i.e., about 10%) are required to achieve acceptable yields at defect rates of $10^{-6} \lesssim \lambda_1 \lesssim 10^{-5}$.

Fault tolerant system design requires analysis of weak points and removal of single points of failure. Table 8.7 summarizes the modules in the cache. Fault tolerance type, device count estimates, and simulated MADP results are shown for each module, for unprotected *chip kill* logic, and for the entire cache.

The device requirements of the design are approximately 15 times that of an equivalent non-fault tolerant cache. Thus, if an alternative device technology such as the single electron transistor can be made at least 15 times smaller than current CMOS, it is possible to implement this design in the same chip area as the base design in CMOS.

Figures 8.24 and 8.25 show the simulation results for the improved CAM cache. The fault tolerance of this cache architecture is significantly higher than either of the

Table 8.7: FDT (Cache C) Module MADPs (Simulated)

Module	Device Estimate			MADP (UC)		
	64KB	512KB	1MB	64KB	512KB	1MB
BIST/R or Control		253,756			7.943×10^{-6}	
Control ECC Enc		5,664			7.943×10^{-5}	
Control ECC Dec		7,620			6.31×10^{-5}	
Main ECC Encs		33,984			1.995×10^{-5}	
Main ECC Decs		91,440			1.259×10^{-5}	
Tag ECC Encs		28,320			2.512×10^{-5}	
CSR †	81.2K	649K	1.30M	3.162×10^{-4}	1.778×10^{-4}	1.778×10^{-4}
Control Bus * †	180K	2.89M	5.76M	2.512×10^{-5}	7.079×10^{-6}	3.981×10^{-6}
Data Bus * †	5.09M	81.3M	163M	3.981×10^{-5}	8.913×10^{-6}	4.467×10^{-6}
CAM Array †	66.38M	574M	1.148B	3.981×10^{-6}	3.548×10^{-6}	3.162×10^{-6}
Chip Kill	15,880	15,982	16,016	6.64×10^{-6}	6.60×10^{-6}	6.58×10^{-6}
Entire Cache (CL)†	67.23M	576M	1.151B	1.59×10^{-6}	1.78×10^{-6}	1.59×10^{-6}
Entire Cache (UC)†	67.23M	576M	1.151B	4.47×10^{-6}	3.98×10^{-6}	3.98×10^{-6}

†: 10% spare rows

*: R = 8 for 64KB cache, R=16 for 512KB/1MB

previous architectures. The MADP for a 90% cache yield is greater than 10^{-6} for all three cache sizes. The simulation results are in agreement with the analytical results.

8.8 Conclusions

8.8.1 Comparison to Other Caches. The MADP figures for the three cache architectures are summarized in Table 8.8. The maximum allowable defect probability for the FDT cache design (i.e., Cache C) is three orders of magnitude greater than the either of the other two architectures. As shown, defect clustering does not have a large impact on yield because the focus is on the region where yield is greater than 0.9, where the difference between clustered and nonclustered results is small. In many cases, due to the probability distribution functions of the Poisson and negative binomial models, yield is initially higher for the unclustered Poisson case than for the clustered case.

8.8.2 Hardware Cost Comparison. The great increase in fault tolerance comes at the cost of significant redundancy. Total device counts and redundancy versus the non-fault tolerant architecture are shown in Table 8.9. The level of redundancy required for the improved cache design is large, but grows very slowly as cache size

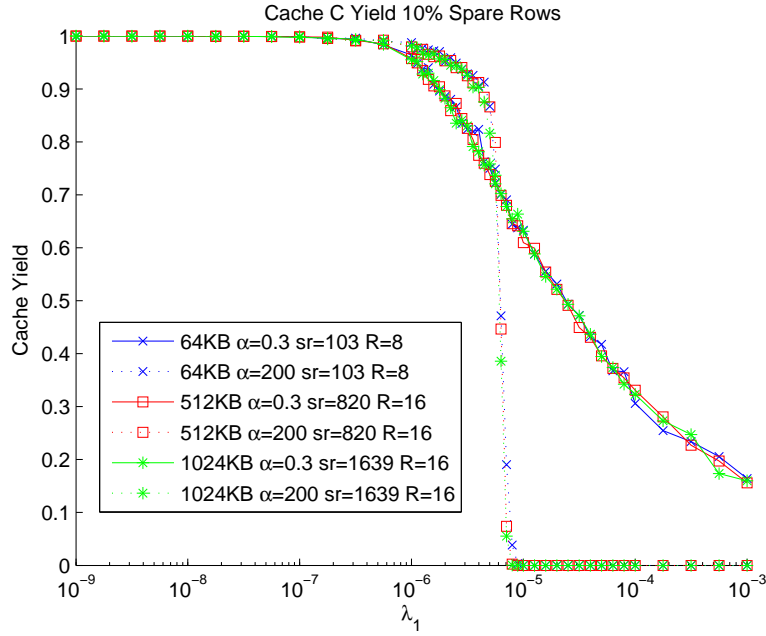


Figure 8.24: The final simulated yields for the improved CAM-based cache provide significant protection against defects.

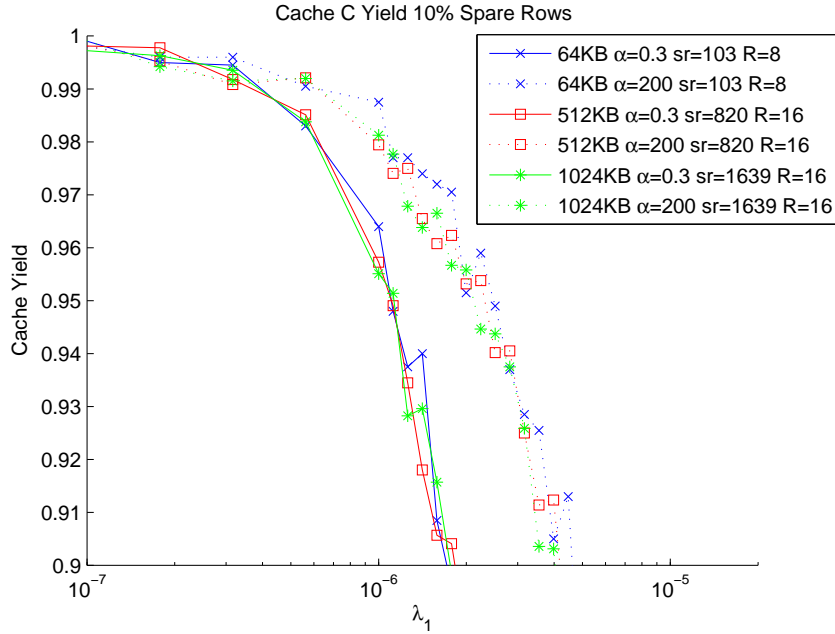


Figure 8.25: A closer view of the simulated yields for the Cache C. The Maximum Allowable Defect Probability to achieve a yield of 90% is better than $\lambda_1 \geq 10^{-6}$ for all three cache sizes.

Table 8.8: Final MADP Results

Cache	FT	64KB	512KB	1MB
A (CL)	No FT	2.53×10^{-8}	3.16×10^{-8}	1.60×10^{-9}
A (UC)	No FT	2.22×10^{-8}	2.87×10^{-8}	1.44×10^{-9}
B (CL)	SR=10%	4.90×10^{-8}	6.40×10^{-9}	3.10×10^{-9}
B (UC)	SR=10%	4.10×10^{-8}	5.00×10^{-9}	2.5×10^{-9}
C (CL)	SR=10%	1.59×10^{-6}	1.78×10^{-6}	1.59×10^{-6}
C (UC)	SR=10%	4.47×10^{-6}	3.98×10^{-6}	3.98×10^{-6}

Table 8.9: Total Device and Redundancy Requirements

Cache		A	B	C
64KB	Total FETs	4.74×10^6	3.14×10^7	6.72×10^7
	Redundancy	1	6.63	14.19
512KB	Total FETs	3.69×10^7	2.28×10^8	5.76×10^8
	Redundancy	1	6.19	15.61
1MB	Total FETs	7.35×10^7	5.02×10^8	1.15×10^9
	Redundancy	1	6.84	15.66

increases. Thus, to compete with the conventional cache architecture implemented in CMOS, an unreliable nanotechnology must be at least 15 times smaller to implement a cache in the same area. Technologies with defect rates smaller than 10^{-6} require less redundancy, and the required size decrease is smaller as well.

8.8.3 SEU Performance and Operational Reliability. In addition to improving manufacturing yield, this architecture offers protection against SEUs occurring in the registers as well as permanent failures that develop during operation. The BIST/R module reserves one bit of error correction capability per 24 bit ECC codeword in the data and tag registers. Codewords that have no faults in the component registers or associated bus logic apply the full correction capability ($t = 3$) to correct SEUs. The cache uses a modified least-recently used replacement strategy, aging cache blocks are eventually flushed, hopefully before more SEUs accumulate than can be corrected.

A final advantage of this architecture is its capacity degrades gracefully as hard faults accumulate in the CAM array. Hard faults can be detected through the occurrence of false-hits and periodic self-testing. The impact of failing devices is initially

masked through ECC and reconfiguration, and later by marking the CAM row faulty and removing it from use. Thus, larger numbers of spare rows ensures a longer useful service life.

8.8.4 The Big Picture. This chapter presented the design for a fault and defect tolerant cache. The design provides a 90% yield for device technologies with $\lambda_1 \geq 10^{-6}$, three orders of magnitude higher than conventional silicon CMOS. The design incorporates the entire cache, including the support circuitry. Although not modelled, the design provides significant protection from SEUs.

This chapter partially answers the question: “How reliable must future device technologies be to produce circuits that compete with modern CMOS?” The design makes no special assumption it is possible to implement majority gates or other fault tolerance logic any more reliably than the rest of the devices in the design. If the design of the “chip kill” devices were optimized, it is possible to improve the MADP of the device technology even further.

This chapter creates the functional architecture for the FDT processor cache. It explores the problem of constructing cache memories resistant to manufacturing defects, device failures, and single event upsets. This analysis incorporates the entire cache instead of just the main memory array. The performance of two typical caches was analyzed, shortcomings identified, and an improved CAM-based cache architecture is proposed that overcomes these limitations. The resulting cache architecture provides a yield of 90% for device technologies with device defect probabilities of 10^{-6} , three orders of magnitude higher than conventional silicon CMOS. The redundancy required to achieve this performance is only 15 times that of the non-fault tolerant design. These results establish targets for viable new device technologies.

IX. Core Logic

9.1 Introduction

This chapter develops the functional architecture for the fault and defect tolerant (FDT) processor. While the previous chapter examined the cache architecture, this chapter examines the core logic of the processor, including the integer and floating point logic, register files, and control logic.

Starting from the hardware cost model for a non-fault tolerant 32 bit CPU, the fault and defect tolerant version is developed. The hardware cost is computed using the techniques in Chapter V. Using this information, a MATLAB[®] simulation analyzes the yield of the fault tolerant design. The maximum allowable defect probability (MADP) for the new design is determined for each cache size and defect clustering case. Finally, yield and redundancy are compared with the initial version. The FDT design meets the yield requirements of Goal 1.

9.2 Methodology

9.2.1 General Approach. The general approach for creating the fault tolerant CPU design is to partition the processor architecture down into discrete modules, which are protected using one or more fault tolerance techniques.

The yield of a fault tolerant chip is determined in part by the number of devices that cannot be protected through redundancy. This typically includes the majority voters, configuration registers, and other hardware used to control redundant modules. As a general approach, the total number of unprotected devices must be minimized. Figure 9.1 shows the reduction in chip yield as the number of devices in the chip kill section increases. For each tenfold increase in N_{ck} , MADP90 decreases by an order of magnitude

Excluding the cache, the FDT processor uses the following module-level fault tolerance techniques:

- TMR

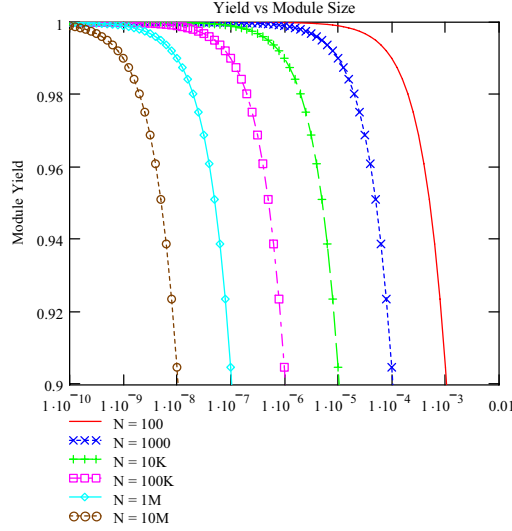


Figure 9.1: The yield of the overall processor is partially determined by the number of devices in the chip kill section.

- Modular Reconfiguration
- TMR-protected reconfiguration

The cache design is from Chapter VIII.

9.2.2 Assumptions. The assumptions from the previous chapter apply:

- All devices have the same probability of failure (i.e., device sizing is not used).
- Majority gates are implemented as Boolean circuits.
- Worst case fault models are used for each module. A fault in the module always disables the module.
- Two fault clustering models are used: the non-clustered (Poisson), and the large-scale clustered (Negative binomial) model [Kor89].

9.3 FDT CPU Design

9.3.1 Initial Architecture. The initial CPU design is a variant of the classic MIPS 32 bit RISC CPU from [PH98]. Detailed hardware models are developed for the MIPS CPU in [MP00] shown in Figure 9.2. The model uses the same device-counting

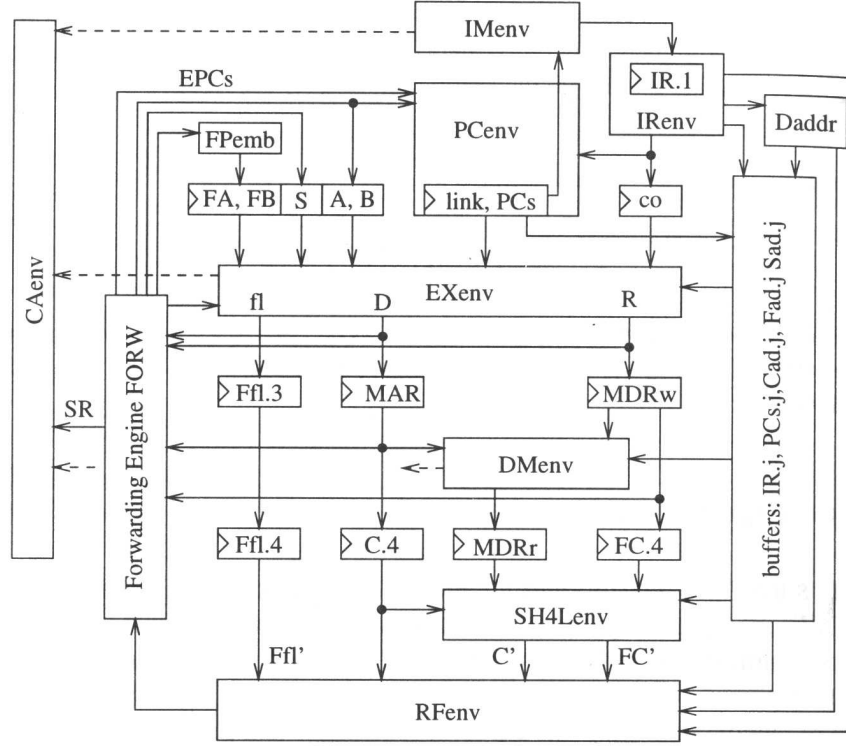


Figure 9.2: Hardware cost models of the MIPS 32 bit CPU were developed in [MP00].

approach proposed in Chapter V. It is adapted directly for use with the FDT design.

The non-fault tolerant hardware cost model is adapted directly from [MP00]. Starting from the bottom up, the mathematical equations for the components are used to develop hardware cost estimates for each module. The results are shown in Tables 9.1 and 9.2. One modification of the module was made: for ease of comparison, the cache memory equations are replaced with the non-fault tolerant cache developed in Chapter VIII. As discussed previously, the total number of devices in the processor is dominated by the cache. Without cache memory, the CPU requires only 290,637 transistors. A further 9.5 million transistors are required to implement dual 64KB caches.

Table 9.1: The non Fault Tolerant 32 bit CPU is made up of the following modules with associated hardware costs (Part One) [MP00].

Module Name	N_{mod}	Description
PCenv	4,692	Program Counter environment
IRenv	602	Instruction Register environment
Daddr	180	
FPemb	788	FP embedding muxes
EXenv	210,600	EXE unit environment
FXU	7,048	Fixed Point Unit
ALUenv	2,812	ALU environment
SHenv	1,932	Integer Shifter environment
Muxes & Drivers	2,304	
FPU	198,940	Floating Point Unit
FCon	2,764	FPU Control
FPunp	12,914	Floating Point Unpacker
FXunp	866	Fixed Point Unpacker
Cvt	4	4
MulDiv	144,980	Multiplier/Divider
SigfMD	142,440	FP Significand Generator
drivers	3,480	
flipflops	10,208	
ROM	1,056	
muxes	690	
4/2 mul	114,108	Multiplier
116 bit adder	3,212	
glue logic	462	
Select FD	9,224	
SignExpMD	1,270	FP Sign & Exponent Generator
SpecMD	112	FP Special Case Handler
flipflops	1,152	
AddSub	9,092	FPU Integer Adder/Subtractor
FXrnd	7,006	Fixed Point Rounder
FPrnd	14,098	Floating Point Rounder
flipflops	2,064	
drivers	5,160	
FPXtr	1,152	Exchange Unit
drivers	3,400	
SH4Lenv	1,720	Shift Left 4 environment

Table 9.2: The non Fault Tolerant 32 bit CPU is made up of the following modules with associated hardware costs (Part Two) [MP00].

Module Name	N_{mod}	Description
RFenv	26,840	Register File environment
GPRenv	8,040	General Purpose Registers
SPRenv	10,434	Special Purpose Regs
FPRenv	8,366	Floating Point Regs
CAenv	1,558	Exception Cause environment
buffers	18,412	
pipeline regs	7,840	
FORW	17,458	Forwarding Unit
SFOR	2,320	Special Purpose Forwarding Unit
FFOR	3,976	Forwarding Unit
CON	11,162	Control Unit
IMC	4	Instruction Memory Controller
DMC	418	Data Memory Controller
MifC	1,300	Memory Interface Control
mux2	18	
I\$ifC	532	Instruction Cache Interface
D\$ifC	750	Data Cache Interface
CE	388	
stall	850	Stall Unit
preCon	2,880	
ConRSR	2,362	Result Shift Regs control
flipflops	224	
glue logic	6	
DivCon	288	Divider Unit control
CON_Mealy	672	Control Unit Mealy state machine
CON_Moore	4,650	Control Unit Moore state machine
Instruction Cache (64 KB)	4.74M	Non-FT cache design from Ch VIII
Data Cache (64 KB)	4.74M	Non-FT cache design from Ch VIII
Total (w/o Caches)	290,637	
Total (w/Caches)	9.77M	

9.3.2 Analytical Model. As developed in Chapter V, chip yield for the unclustered model is the product of the yields of the component modules. Models for cache yield are developed in the previous chapter. The yield expression can be written as

$$Y_{cpu} = \left(\prod_{k \in Mods} Y_k \right) \cdot Y_{icache} \cdot Y_{dcache}. \quad (9.1)$$

Expressions for the clustered defect model are computed using the compounding procedure described previously. In practice, complicated structures contain dependencies that make it impractical to develop analytical models. As before, analytical models are created for the component modules when possible, but the overall CPU yield is found through simulation.

From (9.1), the yield of the chip is bounded above by the most unreliable component. Thus, each module should be analyzed and a fault tolerance technique chosen such that yield and MADP are maximized. In addition, chip yield is bounded above by the yield of the devices unprotected by fault tolerance (i.e., the *chip kill* devices). Thus, CPU yield is

$$Y_{cpu} \leq e^{-N_{ck}\lambda_1}, \quad (9.2)$$

where N_{ck} is the sum of unprotected devices in the entire chip.

The fault tolerant version of the architecture is developed in several steps:

1. Partition the CPU into modules in a logical fashion, attempting to minimize the number of inputs and outputs.
2. Compute the yield curves for each module using TMR, modular reconfiguration, TMR-R, and no fault tolerance.
3. Choose the fault tolerance technique providing the best MADP90.

Partitioning begins at the top level as shown in Figure 9.2. Yield improvement is generally greatest for larger modules. However, the benefit is reduced as the number of outputs is increased (i.e., W_{out}). If a suitable yield cannot be achieved using a large module, the module is examined at a lower level, breaking it into multiple components having fewer devices and fewer outputs.

The process is continued until no further benefit is achieved. For very small modules, or those with a very large number of outputs, the highest yield is obtained by leaving the module unprotected. The results of this partitioning are shown in Tables 9.3 and 9.4. The table shows the hierarchical modules defined in [MP00], number of output lines in the module (W_{out}), and the number of devices in the module (N_{mod}). The tables also show the selected fault tolerance technique, and the number of devices that contribute to the *chip kill* total. As shown on the last line of Table 9.4, the total number of chip kill devices is $N_{ck} = 89,400$ for a CPU with dual 64 KB caches. From Figure 9.1, MADP90 can be initially estimated as $MADP90 \approx 10^{-6}$.

9.4 Yield Performance

9.4.1 Simulation Description. Yield of the FDT processor was estimated by Monte Carlo simulation in the same manner as the cache in Chapter VIII. First, the number of devices in each module are estimated. The total number of devices in the CPU is calculated. The simulation then randomly generates the number of defects in the chip using the appropriate clustering model, and distributes them uniformly among the modules. Each module is examined to determine if the fault tolerance hardware can correct the defect and the functional status of the chip is determined. Repeating this process, average yield is computed. In addition, the 90% confidence intervals for yield are computed as described earlier. Most simulations ran for 2000 iterations. This was sufficient to obtain average yields within 2% of the mean with 90% confidence.

CPU yield is calculated for four cache sizes: no cache, 64KB, 512KB, and 1MB. Data and instruction caches are always the same size. Two clustering parameters

Table 9.3: The Fault and Defect Tolerant CPU is made up of the following modules. N_{mod} is the number of devices in each component module, not the final FT protected module(Part One).

Module	W_{out}	FT Type	N_{mod}	N_{ck}
PCenv	160	Reconfig (R=3)	4,692	2,024
IRenv	301	TMR	602	384
Daddr		No FT	180	180
FPemb		No FT	788	788
EXenv				
FXU				
ALUenv	32	Reconfig (R=3)	2812	488
SHenv	32	TMR	1,932	384
Muxes & Drivers		No FT	2,304	2,304
FPU				
FCon	7	Reconfig (R=3)	2,764	188
FPunp	60	Reconfig (R=3)	12,914	824
FXunp	34	Reconfig (R=3)	866	512
Cvt		No FT	4	4
MulDiv				
SigfMD				
drivers		No FT	3,480	3,480
flipflops	638	Reconfig (R=2)	10,208	5,174
ROM		No FT	1,056	1,056
muxes		No FT	690	690
4/2 mul	58	TMR-R (R=7)	114,108	1,748
116 bit adder	116	Reconfig (R=3)	3,212	1,496
glue logic		No FT	468	468
Select FD	58	Reconfig (R=3)	9,224	800
SignExpMD	14	Reconfig (R=3)	1,270	272
SpecMD	5	TMR	112	60
flipflops		No FT	1,152	1,152
AddSub	71	Reconfig (R=3)	9,092	956
FXrnd	69	Reconfig (R=3)	7,006	932
FPrnd	69	Reconfig (R=3)	14,098	932
flipflops		No FT	2,064	2,064
drivers		No FT	5,160	5,160
FPXtr		No FT	1,152	1,152
drivers		No FT	3,400	3,400
SH4Lenv		No FT	1,720	1,720

Table 9.4: The Fault and Defect Tolerant CPU is made up of the following modules. N_{mod} is the number of devices in each component module, not the final FT protected module(Part Two).

Module	W_{out}	FT Type	N_{mod}	N_{ck}
RFenv				
GPEnv	64	Reconfig (R=3)	8,040	872
SPEnv	32	Reconfig (R=3)	10,434	488
FPEnv	128	Reconfig (R=3)	8,366	1,640
CAenv	32	Reconfig (R=3)	1,558	488
buffers	765	TMR	18,412	9,180
pipeline regs	490	Reconfig (R=2)	7,840	4,760
FORW				
SFOR	128	Reconfig (R=3)	2,320	1,640
FFOR	130	Reconfig (R=3)	3,976	1,664
CON				
IMC		No FT	4	4
DMC		No FT	418	418
MifC				
mux2		No FT	18	18
I\$ifC	11	Reconfig (R=3)	532	236
D\$ifC	18	Reconfig (R=3)	750	320
CE	3	TMR	388	36
stall		No FT	850	850
preCon				
ConRSR	90	Reconfig (R=3)	2,362	1,184
flipflops		No FT	112	112
glue logic		No FT	6	6
DivCon		No FT	288	288
CON_Mealy	13	Reconfig (R=3)	4,650	260
CON_Moore		No FT	672	672
Instruction Cache		Special†	varies	11,680
Data Cache		Special†	varies	11,680
Total				89,400

†: The design from [MP00] is replaced by that from Chapter VIII.

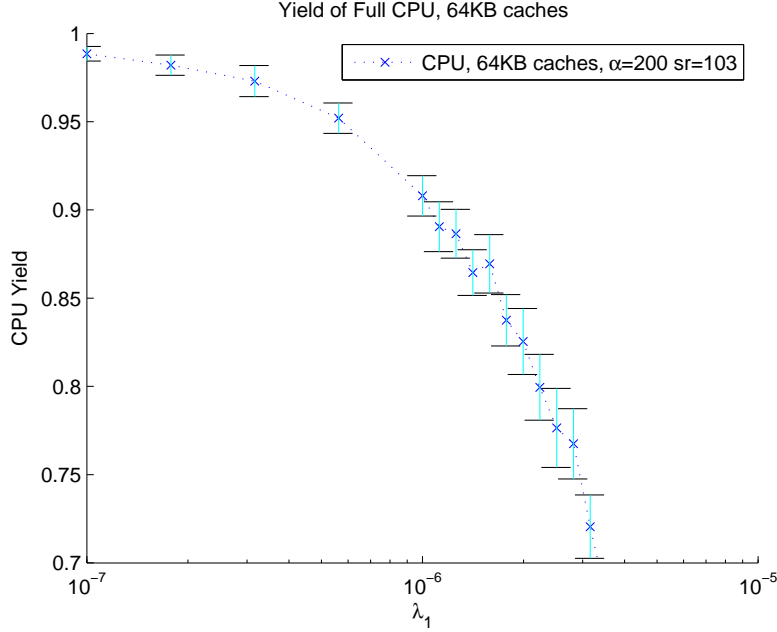


Figure 9.3: Yield of the FDT CPU, with dual 64KB caches. Ninety percent confidence intervals are shown. MADP70 is approximately 3×10^{-6} , two orders of magnitude better than the non fault tolerant design.

were used: $\alpha = 0.3$ is used for clustering, while the Poisson (unclustered) distribution was approximated using $\alpha = 200$.

9.4.2 Simulation Results. The yield of the FDT CPU with dual 64KB caches and no defect clustering is shown in Figure 9.3. Plots for the other cache size and clustering combinations are similar. MADP is determined by the value of λ_1 where the yield crosses the target yield, either 90%, 80% or 70%.

Yield for the FDT CPU core (without caches) is shown in Figure 9.4. Use of the clustered defect model results in higher yield than the unclustered model as defect rate increases. However, in the range of interest (i.e., above 70%) the difference between the two models is small. Thus, the Poisson model can often be used to quickly approximate yield even when the device technology exhibits defect clustering.

The yields of all six cases is shown in Figure 9.5. Upon initial observation, CPU yield does not seem to be dependent on cache size as the curves are very similar. This

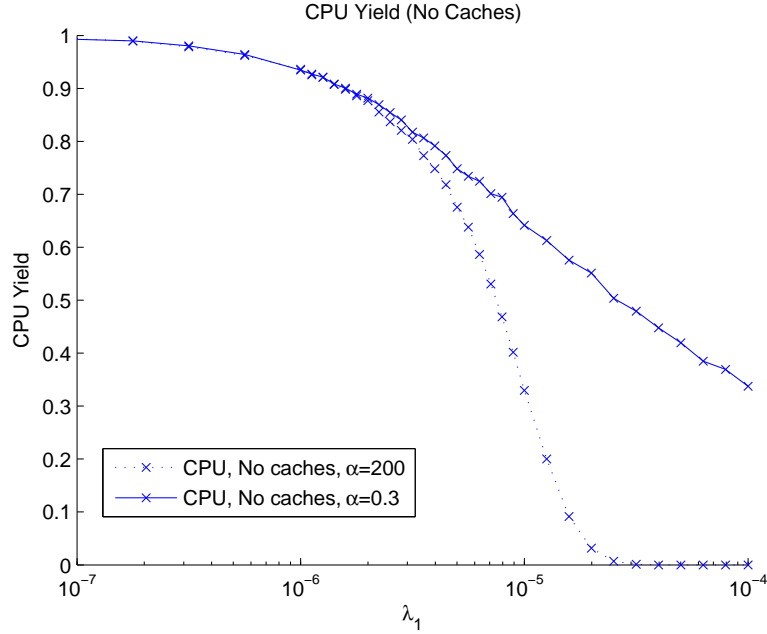


Figure 9.4: Yield of the FDT CPU core, not including caches. For yields above 70%, the two models produce similar values for MADP.

is due to two effects. First, cache yield does not vary much with cache size. Yield is dominated by the number of devices in the chip kill regions of the cache rather than the redundant blocks. Figure 9.6 shows the yields for all six trials on the same axes.

A second effect is the yield for the overall CPU is determined mainly by the CPU core. Figures 9.7, 9.8, and 9.9 show the yields for the six cases. The solid blue lines indicate the FDT CPU yield including both the core and the caches. The dashed green lines indicate the yield of a single cache by itself. The dotted red line with 'x' symbols indicates the yield of the CPU core, which is independent of cache size. For the unclustered model, CPU core yield is less than the yield of a single cache memory. For the clustered model, the yield of the core is very close to that of a single cache. Combined yield is less than either of the two components.

The fault and defect tolerance techniques used in the CPU core were effective in matching the yield performance of the fault tolerant cache. Little benefit would

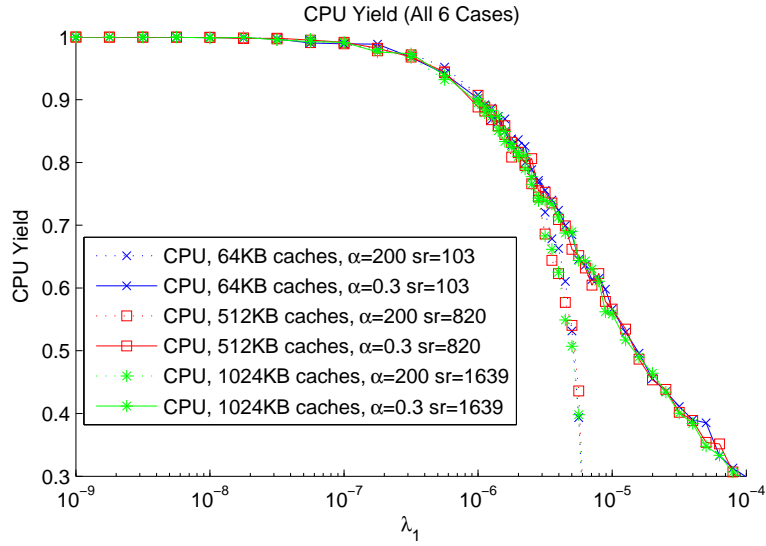


Figure 9.5: The yield for the FDT CPU initially seems independent of cache size. While the cache yield does not vary greatly depending on cache size, the CPU core is less reliable than the caches. This reduces the difference between the curves even further.

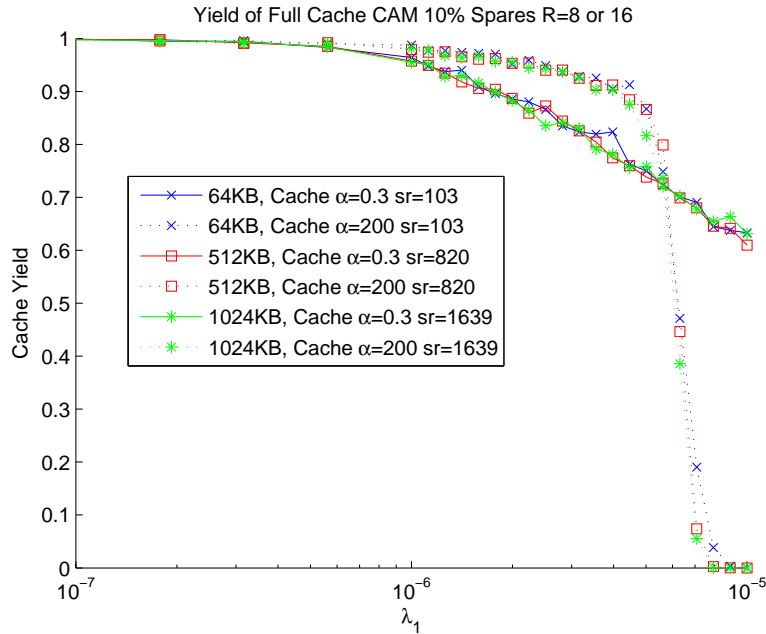


Figure 9.6: The yield of the caches is approximately equal for all cache sizes. This is because cache yield is dominated by the devices in the chip kill section of the cache.

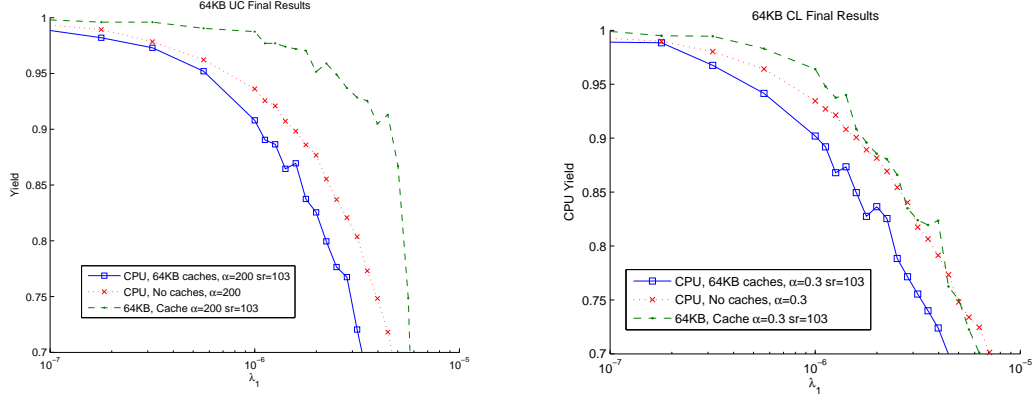


Figure 9.7: Yield for the FDT CPU, including 64KB caches. The plots show the unclustered cases (left) and clustered (right). Combined CPU yield, shown with a blue solid line, is less than either the CPU core (red x's) or a single cache by itself (green dashes).

be achieved through further improvement of the CPU core, as the overall CPU yield would quickly become limited by the cache. For example,

$$Y_{chip} \approx Y_{core} \cdot Y_{cache}^2. \quad (9.3)$$

If the MADP of the CPU core is increased significantly, $Y_{core} \approx 1$ in the range $10^{-6} < \lambda_1 < 10^{-5}$. Thus, the chip yield is limited by the reliability of the cache memory, or

$$Y_{chip} \approx Y_{cache}^2. \quad (9.4)$$

The current design of the fault tolerant CPU core matches the yield of a single cache very closely for the clustered defect case. The difference is greater for the unclustered model, but still reduces the overall MADP70 by 3×10^{-6} .

The MADP results are summarized in Table 9.5. The FDT processor performs very well in terms of yield. For all six cases (three cache sizes and two clustering models), MADP70 and MADP80 are greater than 1.778×10^{-6} . In real terms, the

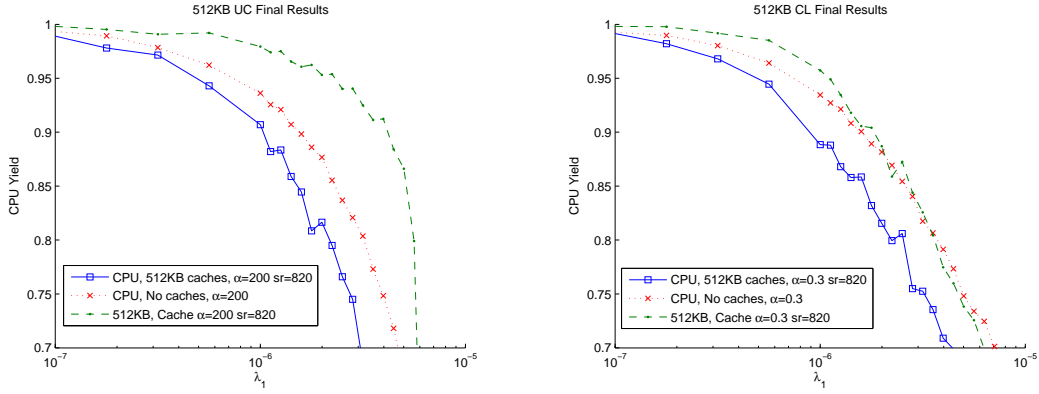


Figure 9.8: Yield for the FDT CPU, including 512KB caches. The plots show the unclustered cases (left) and clustered (right).

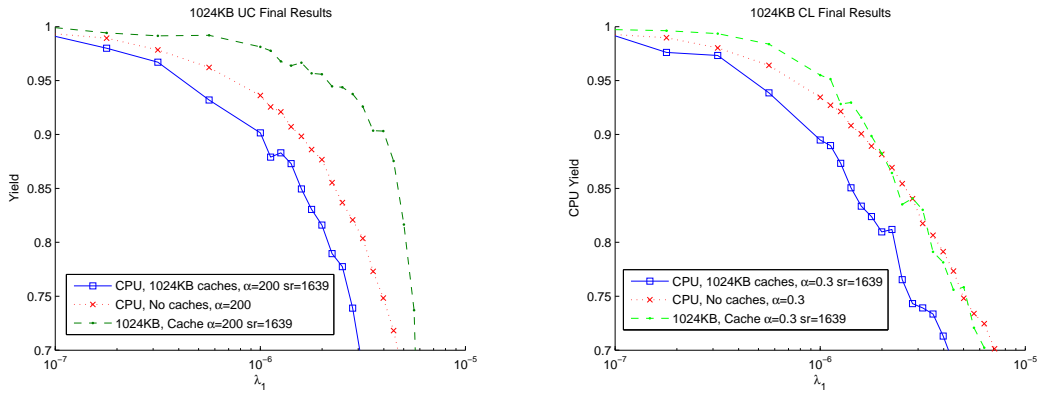


Figure 9.9: Yield for the FDT CPU, including 1MB caches. The plots show the unclustered cases (left) and clustered (right).

Table 9.5: MADP results. Analytical results are shown for the non-fault tolerant design. Simulation results are shown for the FDT CPU.

CPU Design	MADP90	MADP80	MADP70
NonFT, 64KB caches (UC)	1.078×10^{-8}	2.291×10^{-8}	3.649×10^{-8}
NonFT, 64KB caches (CL)	1.289×10^{-8}	3.390×10^{-8}	6.982×10^{-8}
NonFT, 512KB caches (UC)	1.424×10^{-9}	3.015×10^{-9}	4.819×10^{-9}
NonFT, 512KB caches (CL)	1.704×10^{-9}	4.462×10^{-9}	9.262×10^{-9}
NonFT, 1MB caches (UC)	7.173×10^{-10}	1.518×10^{-9}	2.425×10^{-9}
NonFT, 1MB caches (CL)	8.576×10^{-10}	2.238×10^{-9}	4.641×10^{-9}
FDT Core Only (UC)	1.413×10^{-6}	3.162×10^{-6}	4.467×10^{-6}
FDT Core Only (CL)	1.585×10^{-5}	3.538×10^{-6}	7.079×10^{-6}
64KB Cache (UC)	3.981×10^{-6}	5.012×10^{-6}	5.623×10^{-6}
64KB Cache (CL)	1.585×10^{-6}	3.981×10^{-6}	6.310×10^{-6}
512KB Cache (UC)	3.981×10^{-6}	5.012×10^{-6}	5.623×10^{-6}
512KB Cache (CL)	1.778×10^{-6}	3.548×10^{-6}	6.310×10^{-6}
1MB Cache (UC)	3.548×10^{-6}	5.012×10^{-6}	5.623×10^{-6}
1MB Cache (CL)	1.585×10^{-6}	3.548×10^{-6}	6.310×10^{-6}
Core & Dual 64KB (UC)	1.00×10^{-6}	1.985×10^{-6}	3.162×10^{-6}
Core & Dual 64KB (CL)	1.00×10^{-6}	2.239×10^{-6}	3.981×10^{-6}
Core & Dual 512KB (UC)	1.000×10^{-6}	1.778×10^{-6}	2.818×10^{-6}
Core & Dual 512KB (CL)	5.623×10^{-7}	1.995×10^{-6}	3.981×10^{-6}
Core & Dual 1MB (UC)	1.000×10^{-6}	1.995×10^{-6}	2.818×10^{-6}
Core & Dual 1MB (CL)	5.623×10^{-7}	1.995×10^{-6}	3.981×10^{-6}

FDT processor can be fabricated with devices with a failure rate of higher than 10^{-6} and still obtain yields of greater than 80%. Compared to the non fault tolerant design in the top six rows, maximum allowable defect probabilities increases by two to three orders of magnitude.

The hardware overhead required to achieve this yield increase is moderate. Table 9.6 shows the hardware costs for the original and FDT processor designs. The FDT CPU core requires 4.47 times more devices than the original design. The redundancy requirement for the caches is greater, as discussed in Chapter VIII. The FDT CPU design is dominated by the number of devices in the cache, and thus the overall redundancy requirement is between 13.89 and 15.64, depending on the cache size.

Table 9.6: Comparison of the hardware costs of the four different processor configurations. The non-fault tolerant cache design is the design from Chapter VIII rather than the design from [MP00].

Configuration	Non-FT	FDT	Redundancy
No Cache	290,637	1,298,044	$\times 4.47$
Dual 64KB Caches	9.77M	136M	$\times 13.89$
Dual 512KB Caches	74.0M	1.15B	$\times 15.57$
Dual 1MB Caches	147M	2.30B	$\times 15.64$

9.5 Conclusions

This chapter develops the functional architecture of the FDT processor based on the requirements and concept of operations developed in Chapter VII. The analytical models for the fault tolerance techniques discussed in the previous chapters are adapted to create a model for the FDT processor. For comparison, the hardware cost model for a non-fault tolerant CPU was adapted from [MP00]. This model forms the basis of a new hardware cost model for the FDT processor.

The yield of the FDT processor is determined through statistical simulation and shown to meet the requirements in Goal 1. A 32 bit microprocessor can be implemented using device technologies with defect probabilities as high as 1.778×10^{-6} , with a redundancy requirement of approximately 15. This level of redundancy is considerable, but feasible if the new device technology is more than 15 times smaller than silicon CMOS.

X. Device Mapping

10.1 *Introduction*

To this point, the architecture and fault tolerance models examined have been hardware independent. The models operate at the level of Boolean logic gates and higher. Cache and CPU results are based on hardware cost estimates for CMOS technology (e.g., a NAND gate requires four transistors). Further, the models are “switch-based,” and do not incorporate interconnect and other structures. While this is reasonable for CMOS and other technologies in which most failures occur in the switches or can be otherwise evenly distributed between the switches, this approach can produce overly optimistic results for other technologies.

At the same time, adapting the models for a specific device technology may improve the result. For example, if the transistors in the fault tolerance hardware (e.g., majority gates, configuration registers, etc.) can be made larger in a CMOS process, circuit reliability can be improved. Thus, the reliability of a structure is not dependent solely on the number of the transistors, but also their size. In addition, other technologies can implement certain logical functions with fewer devices than CMOS. Thus, fault tolerance hardware may have more efficient implementations and higher yield.

A complete examination requires consideration of the device technology in addition to the logical design of the architecture. This chapter examines the challenges of implementing the proposed fault tolerance techniques onto several emerging successor technologies to silicon CMOS. One technology, QCA, is examined in detail to show how the fault tolerant design should be adjusted from the original “device independent” model. It addresses the three parts of Goal 3 below.

- Goal 3.1: Demonstrate how the proposed architecture may be implemented using a non-CMOS device technology.
- Goal 3.2: Develop a methodology for estimating the hardware area, power consumption, and operating speed.

- Goal 3.3: Determine the minimum performance characteristics of the target device technology necessary to fabricate a processor with the characteristics described in Goal 1.

10.2 *Technology Choices*

The “device independent” model proposed in previous chapters works unmodified for conventional silicon CMOS. Likewise, it can be adapted with little modification to nanotechnologies most similar to CMOS. These technologies should have the following characteristics:

- The most common source of manufacturing defects are in the switching devices (i.e., transistors).
- Other sources of defects, such as interconnect, should be distributable between switching devices such that for any two modules, the ratio between the alternate source and the switching devices is identical. For example, if module A has twice as many transistors as module B, then the probability of failures in the module A interconnect should be twice that of the interconnect in module B. Thus, the parameter λ_1 represents the average probability per device (including its associated interconnect), rather than just the probability the transistor fails in isolation.

Several technologies are well suited for the proposed model:

- Nanoscale CMOS (i.e., sub 100nm process size)
- Single Electron Transistors (SET) [GP98]
- Resonant Tunneling Diode Transistors (RTD) [GP98]

Other technologies are not as well suited for device-based yield modeling. The models used thus far assume defects occur in the interconnect and other structures evenly. This is appropriate for silicon CMOS, but can produce inaccurate results for other technologies. One such technology is quantum dot cellular automata (QCA).

In QCA, the basic unit of construction is the quantum dot cell. These cells are used to construct both logical devices as well as the interconnect. All of these cells can suffer defects and failures that produce errors in operation. Indeed, the vast majority of quantum dots are used to create wires rather than gates. As the length of the wires is dependent on layout as well as the number of switching devices, yield estimates for wires cannot be based simply on the number of switching devices in the module.

A great deal of QCA-related research has been done in recent years. While the technology is far from maturity, it has advanced sufficiently so that simple logical circuits can be designed. From these, it is possible to develop estimates for hardware cost and area. For this reason, QCA is used as the target technology for the remainder of this chapter. The next section provides additional background on QCA. Later, the implementation of fault tolerance techniques for QCA are developed as well as yield expressions and hardware cost.

10.3 QCA Background

10.3.1 QCA Basics. Quantum dot cellular automata were first proposed by Lent and Porod [LTPB93]. As shown in Figure 10.1, the basic unit of construction for QCA is the cell, consisting of four or six quantum dots arranged in a square [GP98]. Two of the four holes contain electrons, in opposite corners. Application of electric charge causes the electrons to shift position to the opposite corners. Thus, it is possible to represent Boolean states by the position of the two electrons in the QCA cell. Information is transmitted by charge instead of current. For use in conventional circuits, it is envisioned that current-based signals entering a QCA block will be converted to logic states by drivers. Likewise, the QCA logic states on the outputs will be converted back to electrical signals by electrometers [AOT⁺99].

If two or more QCA cells are placed close together, the charges interact with each other. The location of the two electrons in one cell can cause a shift in the location of the electrons in the adjoining cells. Potential barriers confine the charges

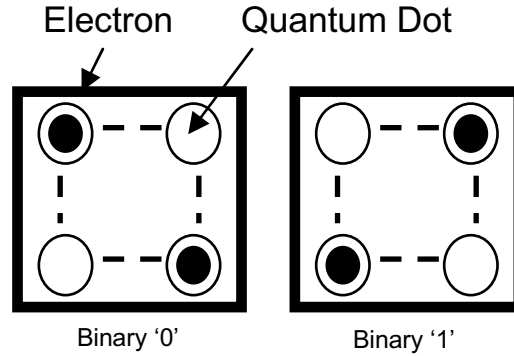


Figure 10.1: QCA logical states are represented by the polarity of the electrons. The left cell represents logic ‘0’, while the right cell represents logic ‘1’ [NK01].

to the quantum dots. To change the location of the electrons, the barriers are reduced through the application of an external electric charge. When the barrier is reduced, the dots in the cell are free to change position based on electrical repulsion with the electrons in the adjoining cells. No charge is transferred, and as a result, QCA has a very lower power consumption.

The largest obstacles to QCA are the inability to operate at room temperature, high defect rates, and slow speed. All of these obstacles may ultimately be overcome. However, to be useful in large logic circuits, speed and operating temperature characteristics must be equal to or better than silicon CMOS. For the remainder of this chapter, it is assumed that operating speed and temperature are not a limiting factor, but instead manufacturability and defect rates.

10.3.1.1 Operation. Logical state is represented by the location of the electrons in the QCA cell (see Figure 10.1). The electrons in the quantum wells are initially held in place by the high energy barriers. Through application of an external electric field, barriers are lowered and the probability of tunnelling from one well to another increases. The electrons move to the lowest energy state, in part determined by the location of the electrons in adjoining cells. By combining cells, logical operations can be performed.

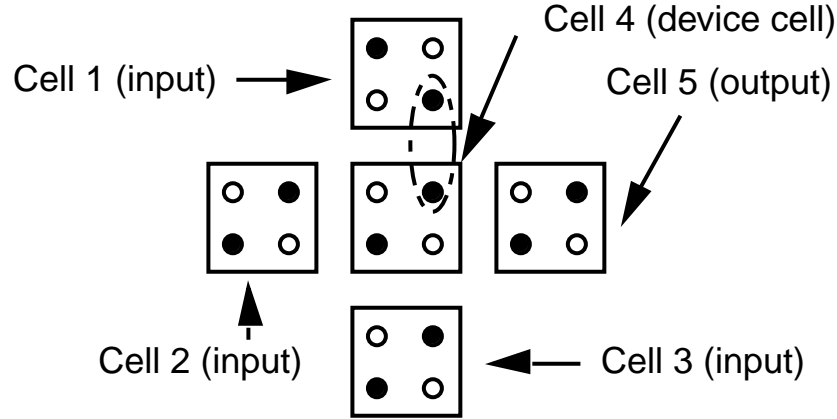


Figure 10.2: Three-input majority gate is implemented in QCA with only five cells [NK01]. The dotted line indicates a repulsion force between two electrons. The electrons in the device cell are held in place by the forces of the electrons in cells 2 and 3.

The basic logic unit in QCA is the three input majority gate shown in Figure 10.2. The top, bottom, and left cells act as the inputs, while the central cell is the computation cell. It assumes the polarization of the majority of the inputs. The output cell on the right assumes this orientation, and the state can be carried to other devices. Note that logical AND and OR gates are simply constructed by tying one of the inputs to ‘0’ or ‘1’, respectively.

One difference between CMOS and QCA is the function of the clock in the circuit [DK03]. In QCA, the clock provides the power to run the circuit, and lowers the energy barriers between the cells to allow state changes. In effect, information is pumped through the circuit by the clock signals.

A four phase clocking scheme is commonly used [HL01], with each phase shifted by 90 degrees. The four clock phases are: switch, hold, release, and relax [NK01].

- Switch phase. Interdot potential barriers are raised so that cells become polarized according to the state of their adjoining drivers.

- Hold phase. Barriers are held high so that the cell remains polarized and fixed in position to act as the input to the adjoining cells.
- Release phase. Barriers are lowered and the cells relax to unpolarized states.
- Relax phase. Barriers remain lowered and cells remain in the unpolarized state.

The clock signals thus control the flow of information through the circuit. One unique aspect of QCA is an inherent latching action during the hold state. Clock signals are provided by external electric fields. One proposal is the use of silicon clock wires embedded in the substrate [HL01, NRK04]. Another proposal uses additional adjoining QCA cells [TL99].

10.3.1.2 Types. Three types of quantum dot devices have been proposed:

- Semiconductor
- Molecular
- Magnetic

Semiconductor QCA is the most common research device, and was first described by Lent and Tougaw in 1994 [LT94]. Semiconductor QCA uses quantum wells formed from aluminum, with tunnel junctions of AlO_x [AOT⁺99]. The cells are approximately 20nm in width. While easier to fabricate than other types, the large size of semiconductor QCA requires extremely low temperatures to operate.

Molecular QCA offers the potential for smaller devices and room temperature operation. It uses redox sites within a molecule and a bridging ligand as the junction between them [LIL03]. Molecular QCA devices have not yet been fabricated, although candidate molecules have been identified [GMI⁺99]. The size of the quantum dot is envisioned as 1-5nm, with a cell size of 10nm [NK04, MHTL05].

A third alternative for QCA is magnetic, but it is currently envisioned as being too slow for use (with switching speeds on the order of only 10kHz [Sem03]).

10.3.1.3 QCA Manufacture. Fabrication of QCA devices is discussed in [AOT⁺99,NK04,NRK04,DK03]. Three building blocks are required: the QCA cells, a substrate upon which they can be attached, and support mechanisms (e.g., clock lines and input/output current drive devices). Proof of concept devices have been constructed [AOK⁺00,Sni98]. For semiconductor QCA, fabrication may be possible using processes adapted from CMOS such as e-beam lithography to define structures [AOT⁺99]. Fabrication begins with a silicon wafer in which silicon clock wires are formed. Next, electron beam lithography cuts trenches for the QCA cells into the silicon. Finally, the wafer is soaked in a bath containing QCA molecules [Lie02,HWLB02].

For molecular QCA, placement could be done by DNA tiles [NK04,NRK04]. These tiles form well defined shapes to which QCA cells can attach. Tiles are combined together to form rafts. Molecular recognition could be used to differentiate locations on the raft for cell attachment, forming arbitrary structures.

10.3.1.4 Area, Power, and Speed. The device density of QCA is much higher than conventional CMOS. A semiconductor quantum cell is approximately 25nm by 25nm [GP98]. Theoretically, this results in a device density of $10^{11}cm^{-2}$, significantly higher than current CMOS. However, since this includes vacant cell locations, usable device density is much lower, and must include cells used for interconnect.

Power dissipation is predicted to be much lower than CMOS, since current does not flow through the devices [GP98,NK01]. Power consumption is roughly $P_{qca} = 10^{-10}$ per device.

Speed of QCA devices is much faster than silicon CMOS. Speed comparisons for several memory designs using QCA are found in [NF01]. Individual cell switching speed for metal semiconductor QCA is $T_{qca} = 2ps$. For molecular QCA, switching speed is estimated at $T_{qca} = 0.02ps$.

10.3.2 QCA Logic Design.

10.3.2.1 Differences from CMOS. In most cases, logical devices and circuits can be implemented in QCA with some modification [WJD03]. The following sections show the implementation of logical building blocks from which large scale circuits can be constructed. The most important differences from conventional CMOS are:

- Wiring is constructed from QCA cells, and must be included in reliability analysis.
- QCA is charge based, and cannot drive conventional electrical circuits. Input and output circuits must be used.
- Complicated clocking is required to move information through the circuit.
- Hardware costs are often very different from the CMOS models proposed in previous chapters.
- QCA possesses some new capabilities, such as the ability for perpendicular wires to cross each other without interference. Likewise, QCA does not possess some basic capabilities, such as the tri-state driver or transmission gate. Other structures must be used instead.

10.3.2.2 Basic Components. The basic logic gate is the majority gate (Figure 10.2). AND and OR gates are formed by fixing one of the inputs at ‘0’ or ‘1’. Unlike CMOS, QCA inverters are more complicated than AND and ORs (Figure 10.3).

A wire crossing is shown in Figure 10.4. By rotating the cells in one wire, perpendicular wire crossings are possible without interference. The rotated cells form an *inverter chain*, which inverts the signal at every cell. One advantage of this approach is that a signal can be split into both its unmodified and inverted form using a single fanout [NK99].

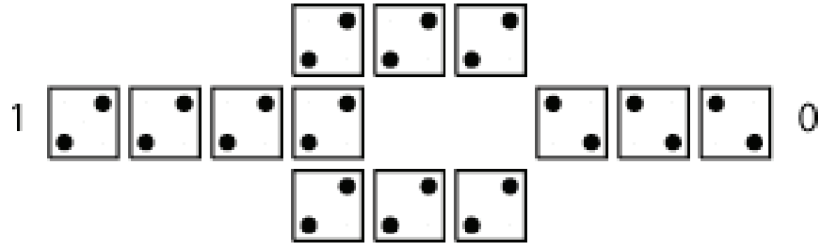


Figure 10.3: QCA Inverter requires more cells than an AND or OR gate [Wal05].

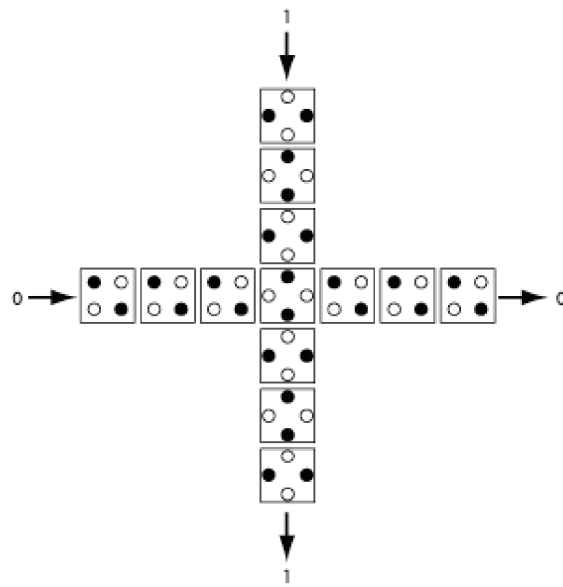


Figure 10.4: QCA wire crossings are possible by rotating the cells in one wire by 45 degrees [Wal05].

10.3.2.3 Memories. Memory implementation is more complex in QCA. Limited memories, such as those useful for pipelined logic, are provided automatically by the wires themselves, due to the use of the four stage clocks. Longer term storage is more complicated. Since cells polarize and depolarize according to the external clock, the ‘memory bit’ must constantly stay in motion. One design is the ring structure proposed in [OVLP05]. A design for an RS flip-flop is proposed in [MHL05]. An SRAM cell design is in [NF01]. Other memory designs are compared in [OVLP05, WVJD03]. In general, all of these designs require more hardware (QCA cells) than the equivalent CMOS design (transistors).

10.3.2.4 Other Structures. Tri-state drivers and transmission gates do not exist in QCA. Bus structures are still possible, however, through the use of an OR array as shown in Figure 10.5. Each output signal is ANDed with an enable line. This signal is ORed with all of the other outputs.

QCA architecture design is still in its early stages, and most designs are relatively small. Designs for XOR gates, simple adders, and other basic devices have been proposed [MTHL04, TL94, WWJ03]. A simple FPGA was described in [WVJD03]. Simple 12 is a very basic processor, consisting of little more than a 12 bit adder, an 8-bit memory, and associated data path [NK99]. A 4 bit CPU was described in [WMSJ05]. Most of these devices are simple “proof of concept” devices implemented in simulation only, and thus not fabricated. They are useful for the development of hardware cost models. These models are discussed in later sections.

10.3.3 QCA Defects. To model the yield and fault tolerance performance of QCA circuits, device failures modes must be considered. This section examines two areas: how QCA devices may fail, and how they are tested.

10.3.3.1 Failure Modes. QCA devices are subject to both soft and hard errors. Soft errors can be caused by particle strike or incorrect design. Line lengths are limited by the ability of the signal to propagate correctly from cell to cell.

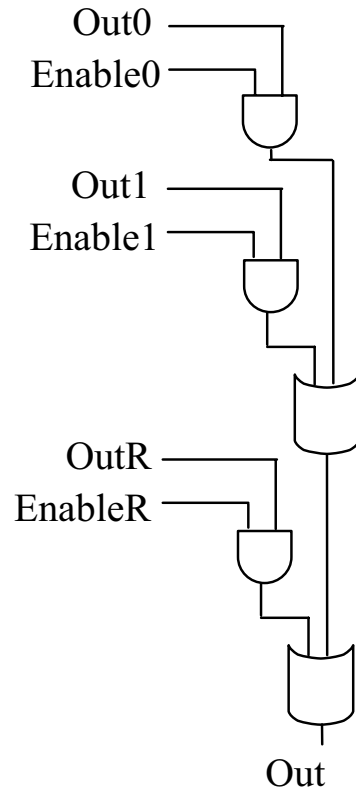


Figure 10.5: Bus structures are possible in QCA using an OR array. An enable line is AND'ed with the output signal before entering an OR tree. The result is similar to the TGATE-enabled bus used in previous chapters.

A wire containing too many cells in a single clock zone may not propagate the state correctly and induce a soft error [LT97]. Parasitic capacitance and crosstalk can also cause problems between wires placed too closely [DK03].

Hard faults are commonly caused by manufacturing defects. Early defect studies were done by [GMI⁺99, Lie02, MHTL05]. Defects can occur in any of the three major components of a QCA chip: the QCA cell logic, the support clock wiring, and the input/output drive circuits. Most research focus has been on the QCA cells. Fabrication of individual QCA cells can be done with fairly low defect rates, on the order of 10^{-5} [JLG⁺03]. For molecular QCA, it is envisioned that working devices can be chemically filtered to remove defective devices prior to deposition onto the DNA substrate. Thus, the most likely defects are likely to be in the deposition phase [THML04]. Although the defect rate is not available due to the immaturity in the technology, it is likely to be in excess of 10^{-5} .

The most likely faults are:

- Shifted cells. A cell placed only a half cell out of alignment can invert a signal and cause it to fail. Thus, placement accuracies for molecular QCA will have to be less than 1 nanometer [DK03].
- Missing cells.
- Extra cells.
- Rotated cells.
- Vertically displaced cells. This can occur if the substrate is not level [NRK04].

Detailed examinations of the effects of these errors on simple QCA device structures are found in [THML04, MHTL05, MOL05]. The most common effects are increased delay, unwanted logical complement (i.e., a vonNeumann error), and logical *stuck at faults*. Clustering of defects has not been examined, and thus the same large scale defect clustering model used for CMOS is assumed for QCA.

10.3.3.2 Testing. Design of test vectors for QCA circuits was addressed in [THML04, TMHL04]. Stuck At Fault testing was shown to be effective at finding most faults in basic QCA circuits and the same techniques applicable for CMOS testing can be used for QCA.

10.3.4 QCA Fault Tolerant Architectures. Given the reliability challenges associated with QCA, several methods have been to provide fault tolerance at the circuit level. Device sizing is possible, particularly for wires. Wider wires and gates should be less subject to logical inversion faults [FT01]. A fault tolerant three-input majority gate gates design is found in [THML04][Wei2005 ref10]. While the majority operation itself is more reliable, interconnect faults are not considered.

Little work has been done thus far on architectural level fault tolerance in QCA. A TMR Shifted Operand (TMR SO) 2-bit adder was proposed in [WWKO05]. This design uses the wire pipelining capability of QCA to latch the inputs and pass it to multiple adders. The output is computed three times and combined with a majority voter. This is similar to the Recompute With Shifted Operands technique described in Chapter II. The technique is useful for the adder circuit, but has limited applicability to other operations.

10.4 Area, Power, and Speed Estimation

Having examined the background of QCA device technology, it is now possible to develop estimators for the area, power, and speed of QCA circuits useful for yield modeling. This section proposes new models for the area, power, and speed of QCA circuits.

10.4.1 Differences from CMOS. The most significant difference between QCA and silicon CMOS is the importance of wires. Since wires must be constructed of QCA cells, the number of cells used in interconnect dominates the number of cells used for the logic devices themselves. Thus, models for area, power, and speed cannot

ignore interconnect. Wiring will therefore be included in all of the models proposed in this chapter.

QCA possesses several unique advantages as well as disadvantages that must be considered for logic design. Some logical gates can be fabricated with fewer resources than in silicon CMOS, while others require more. Transmission gates and tri-state drivers do not exist in QCA, and must instead be implemented with OR arrays.

Clocking circuitry and the input drivers and output electrometers will also consume area and power, and may contribute to defects. Indeed, the minimum feature size of the silicon clock wires contribute the overall area of the design of the ALU in [NK04]. Considering only the QCA cells in the core, this design requires $0.55\mu m^2$ of area. If the silicon wires cannot be made as small as the QCA cells, the entire design must expand to the size of the wires. In this case, the design expanded to $13.9\mu m^2$.

It is possible solutions will be found for the clock wiring size problem. If the clock signal is distributed by QCA wires rather than silicon, the circuit may be made much smaller, on the scale of the QCA devices themselves.

Estimation of QCA characteristics uses a node-based model as shown in Figure 10.6. A similar approach was first used for defect characterization in [MHL05]. In addition to considering devices used in the logic gates themselves, the new QCA model adds wiring structures. The following wire structures are used:

- Wire segment
- Wire corner (i.e., L-shaped)
- Wire fanout
- Wire crossing (i.e., intersection of two wires)

10.4.2 Area Estimation. The QCA chip will be composed of three modules:

- Input drivers to convert external current signals to QCA charge signals.

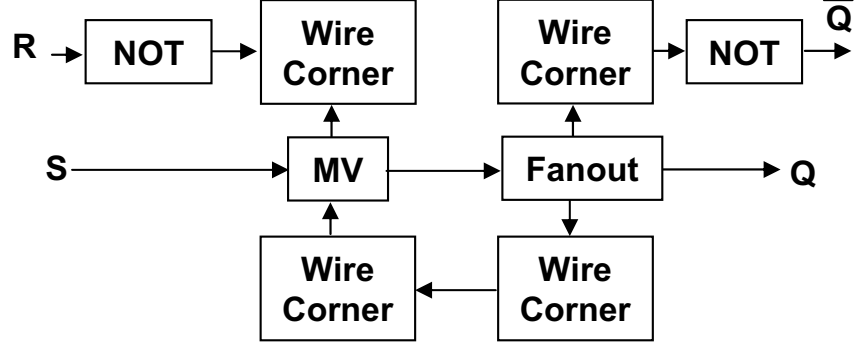


Figure 10.6: Hardware estimation for QCA can be done using a node-based model similar to that used in SPICE. For QCA, wire structures must also be included [MHL05].

- Output electrometers to read the charge state of the QCA logic and convert to current based signals used externally.
- QCA logic.

Thus, total area is defined as:

$$A_{chip} = A_{drivers} + A_{electrometers} + A_{qca_core} \quad (10.1)$$

The driver and electrometer circuits are silicon CMOS and will not be considered further. As stated in the previous section, the area of the QCA core may be limited by the minimum feature size of the clock network. The QCA core area is

$$A_{qca_core} = \max(A_{qca}, A_{clock}). \quad (10.2)$$

It should be noted that a large clock distribution network will impact power, speed, and reliability of the QCA core logic. If the circuit size is expanded due to the silicon clock wires, the lengths of the QCA wires increases. Since the wires are composed of more QCA cells (each of which can fail), reliability will decrease. Likewise, power and switching speed will be negatively impacted. If the size of the

Table 10.1: Hardware costs of basic QCA devices.

Device	Cost	Source
MAJ3	5	[Wal05]
AND, OR	5	[Wal05]
Inverter	9	[Wal05]
Wire Fanout	5	[Wal05]
Wire Corner	5	[Wal05]
Wire Crossing	5	[Wal05]
XOR2	263	[MHTL05]
Full Adder	138	[WWJ03]
SRAM-like memory cell	158	[WVJD03]
RS Flip Flop	66	[MHTL05]
D Flip Flop	198	estimated from 3 RS flip-flops

QCA circuit is limited by the minimum size of the clock wires, the circuit is said to be *clock area limited*. Otherwise, the circuit is said to be *cell area limited*.

Table 10.1 shows the QCA cell hardware cost for several basic structures, assuming the design is cell area limited. From these structures, larger circuits can be estimated.

10.4.2.1 Wire Estimation. Wire cost estimation is the most complicated aspect of the hardware cost. The total cost of the wires is

$$C_{wires} = C_{straights} + C_{corners} + C_{fanouts} + C_{crossings}, \quad (10.3)$$

where $C_{straights}$ is the cost of all the straight wire segments, $C_{corners}$ is the cost of the wire corners, $C_{fanouts}$ is the cost of wire fanouts, and $C_{crossings}$ is the cost of all the wire intersections. For all but the straight wires, cost is constant (cf., Table 10.1) multiplied by the number of instances. For straight wire segments, each wire has a different length, or

$$C_{straights} = \sum_{i \in W} C_{wire}(i), \quad (10.4)$$

where W is the set of all the wires in the module, and $C_{wire}(i)$ is the cost of wire i .

While accurate estimation of wire length is possible for small circuits, it is extremely difficult for larger circuits unless the actual layout is available. For fault tolerance analysis, it is helpful to create estimates for wire costs without having to create the actual circuit layouts. Thus, the average wire length can be defined as

$$AVGWIRE = k_{silicon} \cdot \frac{\sum_{i \in W} C_{wire}(i)}{NumWires}, \quad (10.5)$$

where W is the set of all wires, $C_{wire}(i)$ is the cost of wire i , and $k_{silicon}$ is a constant to reflect the scaling required for clock area limited designs (i.e., $k_{silicon} > 1$).

Rather than dealing with thousands of individual wire lengths, the model will be simplified to two types: *short wires*, which are used for all local interconnect, and *long wires*, which depend on module size and are used for intermodule and global connections. Short wires are independent of the size of the module. The hardware cost for a short wire is simply the average wire length.

$$C_{shortwire} = AVGWIRE. \quad (10.6)$$

For long wires, such as those used in TMR to connect the outputs of three modules to the majority voter, length is highly influenced by the size of the module. Length should be included in the hardware cost estimator. Thus, the model for a long wire is defined as

$$C_{longwire}(i) = L_{module}(i) \cdot k_{silicon}, \quad (10.7)$$

where $L_{module}(i)$ is the length of module i (in cells), and $k_{silicon}$ is the silicon scaling constant. $L_{module}(i)$ is estimated by assuming that the module layout is square. Module length is directly dependent on the number of cells in the module, N_{mod} . However, since separation between cells is required, a large number of empty cells are found in the module and must be counted as well. A new parameter, θ_{qca} , is the fraction of

tiles in the module that are occupied by a QCA cell. Values of θ_{qca} depend on the design, but will be between $0 < \theta_{qca} \leq 0.5$. Thus, the estimated cost for a long wire is

$$C_{longwire}(i) = \sqrt{\frac{N_{mod}(i)}{\theta_{qca}}} \cdot k_{silicon}, \quad (10.8)$$

where $N_{mod}(i)$ is the number of devices in module i , and θ_{qca} is the cell fill fraction.

10.4.3 Power Estimation. Power and speed are estimated in a manner similar to hardware cost. Unlike hardware cost, however, empty space does not need to be accounted for. Power and speed are dependent solely on the number of QCA cells in the module.

$$P_{chip} = P_{drivers} + P_{electrometers} + P_{qca_core} \quad (10.9)$$

As before, the drivers and electrometers are estimated using techniques for silicon CMOS, and are not discussed further. For the QCA logic, total power consumption is estimated from the total number of QCA cells:

$$P_{qca_core} = N_{core} \cdot P_{qca}, \quad (10.10)$$

where N_{core} is the number of QCA cells in the design, and P_{qca} is the estimated power consumption of a single QCA cell. From Section 10.3.1.4, $P_{qca} = 10^{-10}W$.

10.4.4 Speed Estimation. In silicon CMOS, the operating speed of a sequential circuit can be found by determining the propagation delay through the critical path of the circuit (i.e., the path with the longest delay). The signal can pass through multiple logic gates in a single clock cycle. From the delay estimate, the maximum operating frequency can be determined. In a typical pipelined CPU, most of the hardware in each stage is combinational, with results being fed to the pipeline registers.

For QCA, the clock plays a more central role. In a sense, all circuits are clocked circuits, since the four stage clock is used to ‘pump’ the information through the circuit. Clocking is required to produce results in the logical gates, as well as move information down wires. Long wires must be split into multiple clock zones to restore the logic level and prevent errors. In addition, information can pass through no more than one logical gate per cycle. Thus, the work accomplished per clock cycle in QCA is likely to be much less than an equivalent CMOS circuit.

One design strategy for QCA clock circuits is a nested pipelining technique in which the conventional pipelined CPU architecture is divided into many smaller clock cycles. Logic flow in a single pipeline stage (e.g., the EXE stage) would be completed in X clock cycles, at which point the results are latched and sent to the next stage. For a Y stage pipeline, total execution of an instruction would require XY clock cycles. Care must be taken in the design of each stage to ensure that all of the results arrived at the pipeline register inputs at the same clock cycle. While this complicates layout of the QCA circuit, it masks the effects of the pipelining from the CPU level architecture, and mitigates negative CPU performance effects from a pipeline that might otherwise be hundreds of layers deep.

Thus, the propagation delay of each pipeline stage should be matched, or

$$T_{IF} = T_{ID} = T_{EX} = T_{MEM} = T_{WB}. \quad (10.11)$$

Delay balancing can be achieved through the careful use of long wires. Addition clock segments would be added to ensure the signals arrive at the pipeline registers at the same time. The overall delay for a CPU pipeline stage, $T_{pipeline}$, is the product of the number of clock zones in the stage, $N_{clockzones}$, and the delay of a single clock zone, $T_{clockzone}$,

$$T_{pipeline} = N_{clockzones} \cdot T_{clockzone} \quad (10.12)$$

The actual clock speed for the QCA cells is determined by the maximum number of QCA cells clocked in a single zone, N_{1zone} and the switching speed of a single QCA cell, T_{sw} , or

$$T_{clockzone} = N_{1zone} \cdot T_{sw}. \quad (10.13)$$

From Section 10.3.1.4, $T_{sw} = 0.02ps$ for molecular QCA. Thus, a single QCA cell can switch at a speed of $5 \times 10^{12}Hz$. As an example, assume that wire length is limited to 25 cells in a single clock zone. Thus, switching time for a single clock zone is $25 \cdot T_{sw} = 5 \times 10^{-13}$, for a maximum clock speed of 2 THz. If a pipeline stage is broken into 50 logic and wire delay zones, the CPU pipeline speed is $2THz/50 = 40GHz$.

10.5 A Defect Model for QCA

This section develops a model for the yield of a QCA module. Yield of a QCA-based chip is

$$Y_{chip} = Y_{qca_core} \cdot Y_{clock} \cdot Y_{drivers} \cdot Y_{sensors}, \quad (10.14)$$

where Y_{qca_core} is the yield of the QCA core logic, Y_{clock} is the yield of the supporting silicon clock circuits, $Y_{drivers}$ is the yield of the input circuits that convert current-based signals to charge-based signals, and $Y_{sensors}$ is the yield of the electrometers that convert output signals back to current-based signals for use off the QCA chip. The clock, driver, and sensor hardware are silicon CMOS based circuits, and can be modelled with conventional techniques.

Yield of a QCA logic module depends on the following factors:

- Number and types of logic structures (i.e., MAJ3, AND, OR, etc.).
- Number, type, and length of wire structures (i.e., Fanouts, wire crossings, long and short wires)

Table 10.2: QCA circuits suffer from several types of defects, each with a different probability

Defect Type	Probability of Module Failure	Probability of Cell Failure	Common Effects
Defective cell	λ_{cell}	λ_{1cell}	VN or SA faults
Missing cell	λ_m	λ_{1m}	VN or SA faults
Displaced cell (lat)	λ_L	λ_{1L}	VN or SA faults
Displaced cell (vert)	λ_v	λ_{1v}	VN or SA faults
Rotated cell	λ_r	λ_{1r}	VN or SA faults
Extra cell	λ_e	λ_{1e}	VN, SA, or wire coupling

- Area of the module layout
- Spacing of unrelated components (i.e., to prevent crosstalk)
- The probability each individual QCA cell is functional.

As discussed in Section 10.3.3.1, QCA logic can fail due to defective QCA cells or faults in cell placement on the substrate. Table 10.2 lists the common defects. The symbols represent the probability the module suffers a fault of this type.

The probability of each of these faults is directly proportional to the number of cells in the module.

$$\{\lambda_{cell}, \lambda_m, \lambda_L, \lambda_v, \lambda_r, \lambda_e\} \propto N_{mod} \quad (10.15)$$

Unclassified yield of a module composed of QCA cells is thus

$$Y_{qca.mod} = e^{-N_{mod}\lambda_1}, \quad (10.16)$$

where N_{mod} is the number of cells in the module, and λ_1 is the mean probability of defect for a single cell. This probability adds the probabilities of the various types of defects,

$$\lambda_1 = \lambda_{1cell} + \lambda_{1m} + \lambda_{1L} + \lambda_{1c} + \lambda_{1r} + \lambda_{1e}. \quad (10.17)$$

Yield of a module composed of the primitives in the top half of Table 10.1 is

$$Y_{mod} = \prod_{k \in ModTypes} Y_k \quad (10.18)$$

where

$$ModTypes = \{MAJ3s, NOTs, short\ wires, long\ wires, fanouts, wire\ crossings\}. \quad (10.19)$$

The yield for all of the MAJ3 gates is computed as

$$Y_{MAJ3s} = (Y_{MAJ3})^{NumMAJ3s} = e^{-C_{maj3} \cdot \lambda_1 \cdot NumMAJ3s}, \quad (10.20)$$

where C_{maj3} is the number of QCA cells in the MAJ3 gate. Similar expressions exist for the other primitive elements. Thus, the yield of complicated structures is derived from basic building blocks. Yield expressions for QCA versions of TMR, modular reconfiguration, and TMR-protected reconfiguration are derived in the next section.

10.6 Fault Tolerant Circuits for QCA

With some minor modification, the three fault tolerance schemes used in previous chapters are useful for QCA. This section illustrates the design of the circuits and develops mathematical models for yield.

10.6.1 TMR. Triple modular redundancy is easily implemented in QCA. The basic logic gate in QCA is the MAJ3 gate, implemented with only 5 QCA cells. In CMOS, a MAJ3 gate requires 12 transistors. Thus, without regard to wiring, TMR has the potential to perform better on QCA than in CMOS.

10.6.1.1 Circuit. The circuit diagram for QCA TMR looks similar to that of CMOS. The fault tolerance model must include the wire segments. Figure 10.7 shows the QCA node layout of the input wiring.

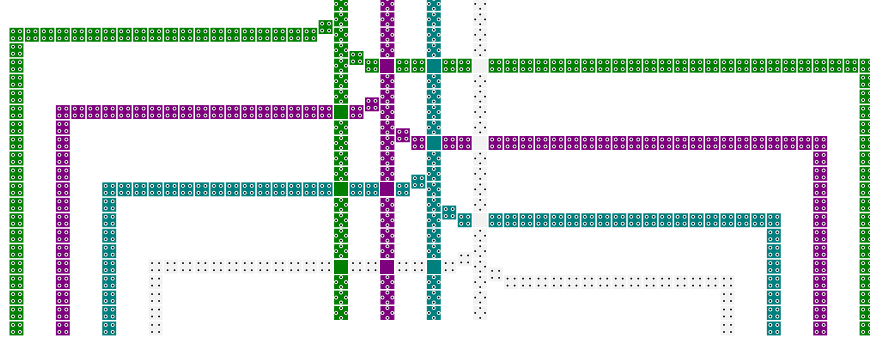


Figure 10.7: QCA 3-Module input layout. This circuit distributes the $W_{in} = 4$ inputs to three modules. A similar layout will also be used for Modular Reconfiguration and TMR-protected Reconfiguration. The colors denote different signals rather than clock zones, which are not shown.

Figure 10.8 shows the layout of the output wires and majority gates. Provided that parallel wires are placed sufficiently far apart (according to the design rules), defects are unlikely to affect multiple wires except at the wire crossings.

10.6.1.2 Yield Expressions. The yield expression for QCA TMR is adapted from the TMR expression from Chapter V, and is

$$Y_{tmr} = Y_{cktmr} \sum_{k=2}^3 \binom{3}{k} Y_{supermod}^k (1 - Y_{supermod})^{3-k}, \quad (10.21)$$

where Y_{cktmr} is the hardware unprotected by redundancy. In this case, it is the input wiring prior to the fanouts, and the output majority gates and wire crossings. $Y_{supermod}$ is the yield of a supermodule. The *supermodule* is defined as the logic module and attached input and output wiring that are protected by a fault tolerance scheme. For TMR, the supermodule contains the input wiring after the fanouts, and all of the the output wiring excluding the wire crossings.

The yield of the chip kill logic is

$$Y_{cktmr} = Y_{maj3}^{W_{out}} \cdot Y_{ckinputwires} \cdot Y_{ckoutputwires}, \quad (10.22)$$

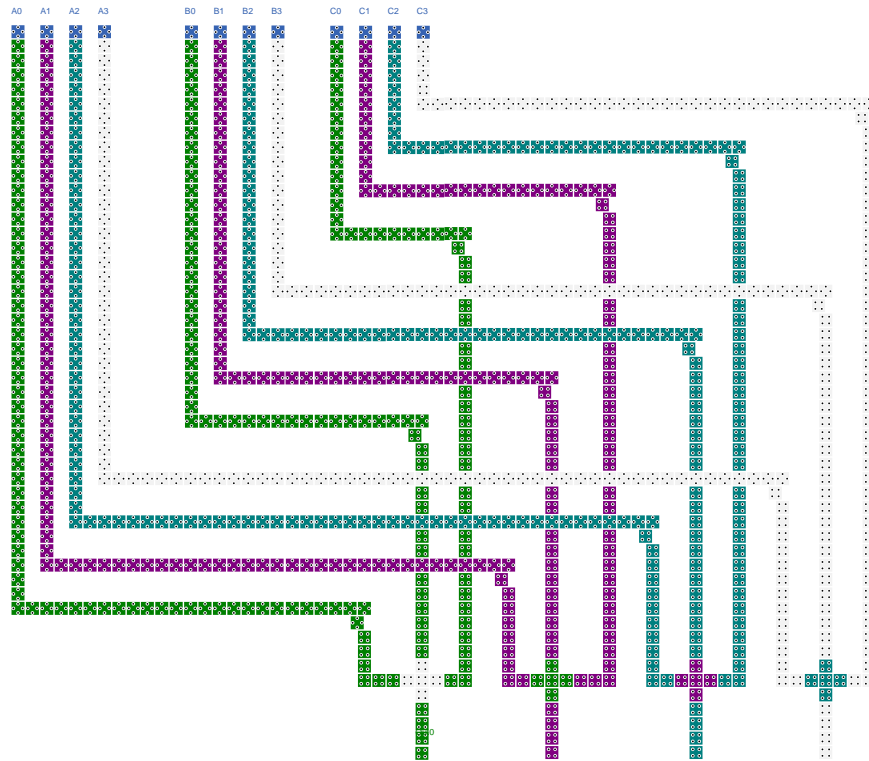


Figure 10.8: QCA TMR Output layout. Color shading indicates different wires rather than clock zones.

where W_{out} is the number of signals in the output of the module, Y_{maj3} is the yield of a single MAJ3 gate, $Y_{ckinputwires}$ is the yield of the input wires prior to and including the fanouts, and $Y_{ckoutputwires}$ is the yield of the output wire crossings. Since a fault in a wire crossing can cause errors in signals going to multiple majority gates, a fault here is considered to be a chip kill event. The design in Figure 10.8 is scalable with W_{out} , and will have $3(W_{out} - 1)(W_{out} - 2)$ wire crossings. The yield is thus

$$Y_{ckoutputwires} = Y_{crossing}^{3(W_{out}-1)(W_{out}-2)} \quad (10.23)$$

for the output wire crossings, and

$$Y_{ckinputwires} = Y_{shortwire}^{W_{in}} \cdot Y_{fanout}^{2W_{in}} \cdot Y_{crossing}^{2(W_{in}-1)(W_{in}-2)} \quad (10.24)$$

for the input wires up to and including the last fanout. This section is composed of one short wire and two fanouts per input wire, and $2(W_{in} - 1)(W_{in} - 2)$ wire crossings. W_{in} is the number of inputs to the module. Note that this is a new parameter, as the yield of CMOS TMR depends only on the number of signals in the output.

As a design rule, the best layout will split the wires as early as possible. All of the wiring after the fanouts becomes part of the supermodules, and therefore has some fault tolerance protection. Wiring prior to the fanouts must be included in the chip kill analysis.

Yield of the supermodule is simply

$$Y_{supermod} = Y_{mod} \cdot Y_{inwires} \cdot Y_{outwires}, \quad (10.25)$$

where Y_{mod} is the yield of the module to be protected, $Y_{inwires}$ is the yield of the input wires after the fanouts, and $Y_{outwires}$ is the yield of the output wires (not including the wire crossings).

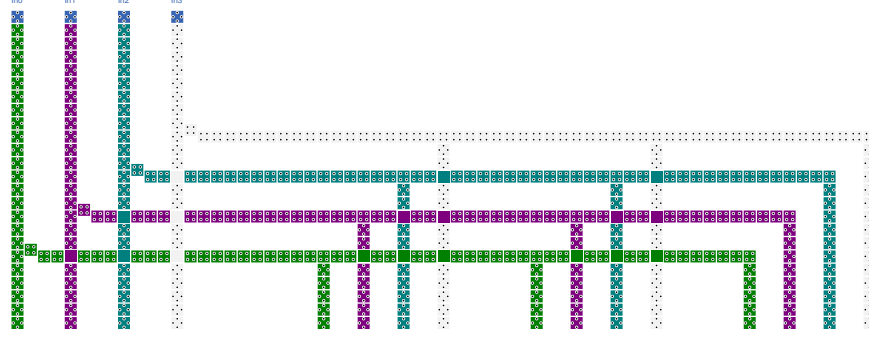


Figure 10.9: QCA R-Module input layout. This circuit distributes the W_{in} inputs to the R modules. This circuit is also used for TMR-protected reconfiguration. Clock zones are not shown.

Each of the input wires consists of one short wire, one long wire, and one corner. Yield of the protected input wires is thus

$$Y_{inwires} = Y_{shortwire}^{W_{in}} \cdot Y_{longwire(N_{mod})}^{W_{in}} \cdot Y_{corner}^{W_{in}}. \quad (10.26)$$

Similarly, the protected output wires are made up of short and long wire segments, as well as several corners. Yield is thus

$$Y_{outwires} = Y_{shortwire}^{2 \cdot 3 \cdot W_{out}} \cdot Y_{longwire(N_{mod})}^{3W_{out}} \cdot Y_{corner}^{2 \cdot 3 \cdot W_{out}}. \quad (10.27)$$

Yield of QCA TMR is compared to conventional CMOS TMR in section 10.7.

10.6.2 Reconfiguration. The circuit for QCA reconfiguration is similar to that of CMOS with some minor modifications. Since TGATES cannot be implemented with QCA, an OR array is used instead.

10.6.2.1 Circuit. The input wire network for modular reconfiguration is shown in Figure 10.9. It is similar to the layout for TMR, with the addition of multiple fanouts to provide the R inputs.

The output network is shown in Figure 10.10. As in CMOS reconfiguration, enable registers are used to connect the module outputs to the common bus. Unlike

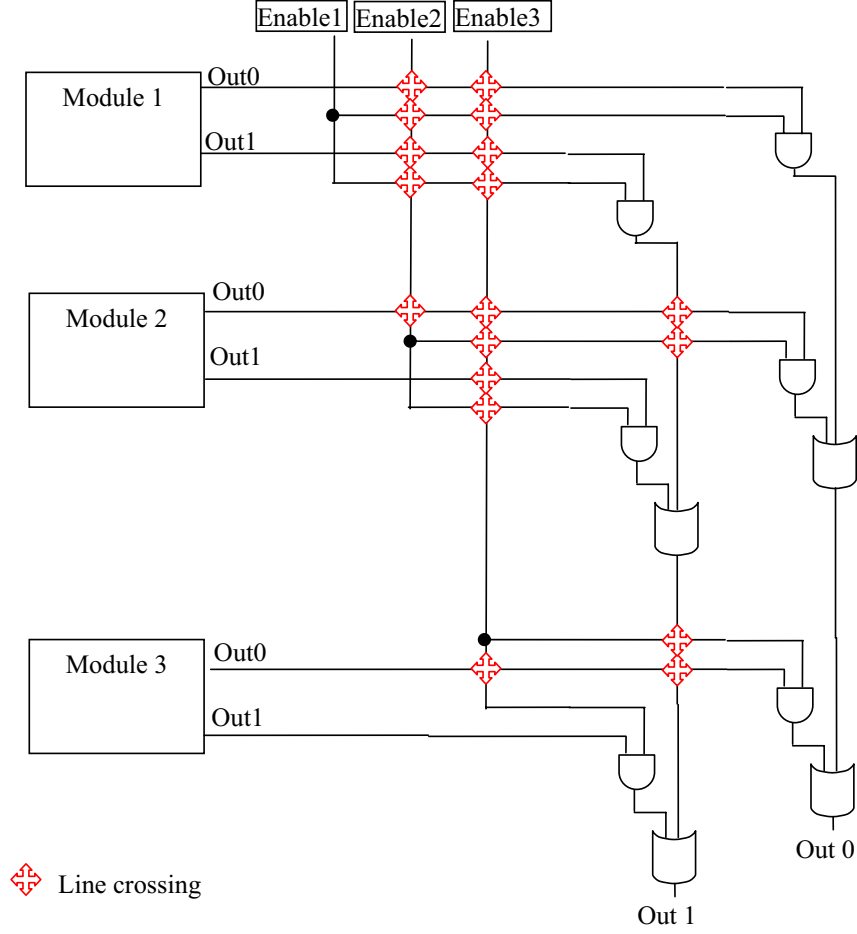


Figure 10.10: QCA Modular Reconfiguration Output Layout. For this example, $R = 3$ and $W_{out} = 2$.

CMOS, which uses TGATES, QCA uses an OR array. Each enable signal is ANDed with the appropriate output. These signals are then combined using $(R-1)$ OR2s to form the output bit.

10.6.2.2 Yield Expressions. Similar to the CMOS implementation, the yield for QCA modular reconfiguration is

$$Y_{reconfig} = Y_{ckreconf} \sum_{k=1}^R \binom{R}{k} Y_{supermod}^k (1 - Y_{supermod})^{R-k}, \quad (10.28)$$

where $Y_{supermod}$ is the yield of the supermodule, and $Y_{ckreconf}$ is the the yield of the non-redundant hardware. This section includes the R configuration registers, $R \cdot W_{out}$ AND2 gates, and $(R - 1)W_{out}$ OR2 gates. This expression is

$$Y_{ckreconf} = Y_{register(1)}^R \cdot Y_{AND2}^{R \cdot W_{out}} \cdot Y_{OR2}^{(R-1)W_{out}} \cdot Y_{mk_inwires} \cdot Y_{mk_outwires}, \quad (10.29)$$

where $Y_{mk_inwires}$ is the yield of the unprotected wires in the input (i.e., the wires prior to the fanouts), and $Y_{mk_outwires}$ models the unprotected output wires.

The yield of the unprotected sections of the input wires includes a vertical short wire segment, a horizontal long wire, and fanouts for each of the $R \cdot W_{in}$ segments. In addition, there are $(W_{in} - 1)(W_{in} - 2)(R - 1)$ wire crossings. All of these segments are included in the chip kill segment, as a failure affects more than one of the modules. The expression becomes

$$\begin{aligned} Y_{mk_inwires} = & (Y_{shortwire} \cdot Y_{longwire(N_{mod})} \cdot Y_{fanout})^{W_{in}(R-1)} \\ & \cdot Y_{crossing}^{(W_{in}-1)(W_{in}-2)(R-1)}. \end{aligned} \quad (10.30)$$

The expression for the unprotected segments in the output logic is

$$Y_{mk_outwires} = Y_{enablewires} \cdot Y_{ORarray}, \quad (10.31)$$

where $Y_{enablewires}$ is the yield of the wires used in the enable logic, and $Y_{ORarray}$ is the yield of the wires used in the remaining OR array. $Y_{enablewires}$ is

$$\begin{aligned} Y_{enablewires} = & Y_{longwire(N_{mod})}^{R(R-1)} \cdot Y_{shortwire}^{2R \cdot W_{out}} \cdot Y_{crossing}^{R(R-1)W_{out}} \\ & \cdot Y_{fanout}^{(R-1)W_{out}} \cdot Y_{corner}^R. \end{aligned} \quad (10.32)$$

Similarly, the expression for $Y_{ORarray}$ is

$$Y_{ORarray} = Y_{shortwire}^{3R \cdot W_{out}} \cdot Y_{longwire(N_{mod})}^{R \cdot W_{out}} \cdot Y_{corner}^{2R \cdot W_{out}}. \quad (10.33)$$

The yield of the supermodule is computed just as in TMR, or

$$Y_{supermod} = Y_{mod} \cdot Y_{inwires} \cdot Y_{outwires}, \quad (10.34)$$

where Y_{mod} is the yield of the module, $Y_{inwires}$ is the yield of the wire segments after the fanouts leading directly into the modules, and $Y_{outwires}$ is the yield of the wire segments exiting the modules prior to the segments in the output chip kill section.

The expression for $Y_{inwires}$ is the same as for TMR, or

$$Y_{inwires} = Y_{shortwire}^{W_{in}} \cdot Y_{longwire(N_{mod})}^{W_{in}} \cdot Y_{corner}^{W_{in}}. \quad (10.35)$$

The expression for the output wires is

$$Y_{outwires} = Y_{shortwire}^{2 \cdot W_{out}} \cdot Y_{corner}^{W_{out}}. \quad (10.36)$$

10.6.3 TMR-Protected Reconfiguration. TMR-protected reconfiguration operates in the same manner as RMR, with a wider output bus to provide three outputs, which are combined using a MAJ3 gate.

10.6.3.1 Circuit. The input section of the circuit is identical to that used for modular reconfiguration in Figure 10.9. The output circuit is shown in Figure 10.11.

10.6.3.2 Yield Expressions. The yield expression for the QCA version of TMR-protected reconfiguration is

$$Y_{tmrr} = Y_{ck_tmrr} \sum_{k=2}^R \binom{R}{k} Y_{supermod}^k (1 - Y_{supermod})^{R-k}, \quad (10.37)$$

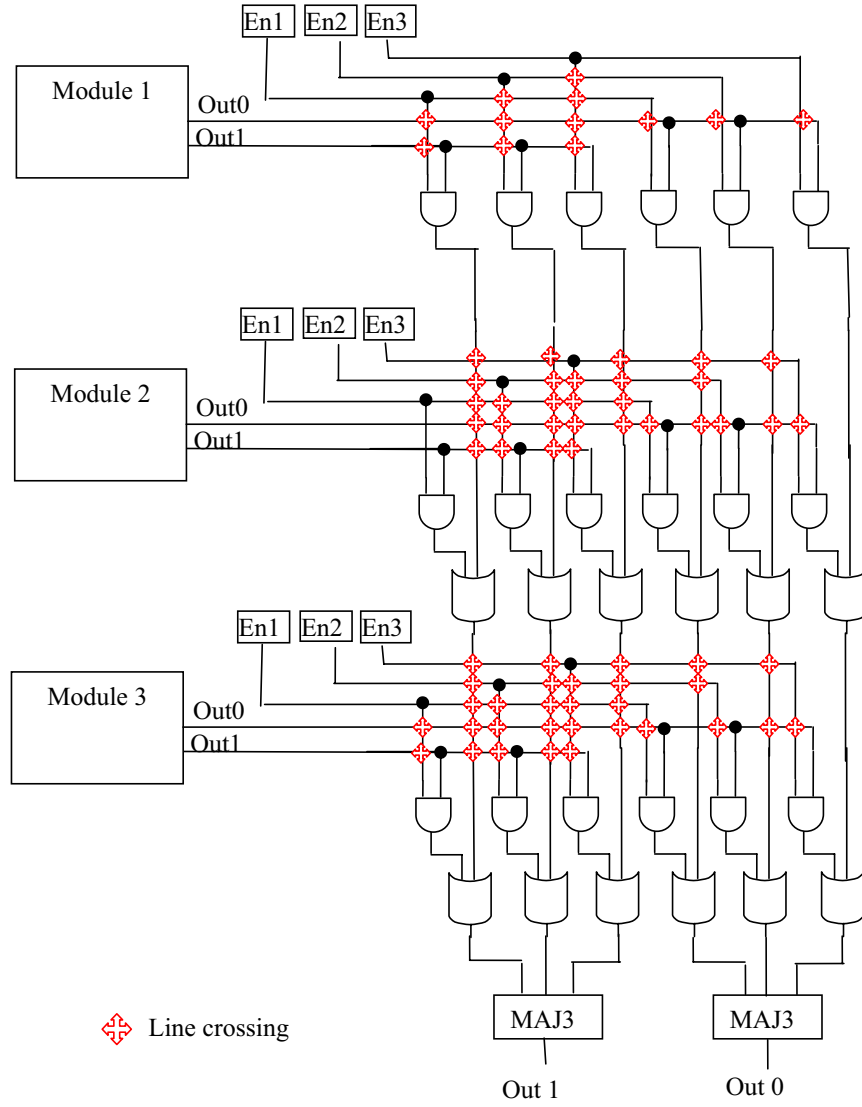


Figure 10.11: QCA TMR-Protected Reconfiguration Output Layout for $R = 3$ and $W_{out} = 2$.

where $Y_{supermod}$ is the yield of the supermodule, and Y_{ck_tmrr} is the yield of the unprotected logic. This section consists primarily of the logic in the output OR array and majority gates. The expression for Y_{ck_tmrr} is

$$Y_{ck_tmrr} = Y_{ORarray} \cdot Y_{maj3}^{W_{out}}. \quad (10.38)$$

For the OR array, at least 2 of the R outputs must be correctable (for all W_{out} bits). The expression is

$$Y_{ORarray} = Y_{mk_ORarray} \sum_{k=2}^3 \binom{3}{k} Y_{1col}^k (1 - Y_{1col})^{3-k}, \quad (10.39)$$

where $Y_{mk_ORarray}$ is the yield of the unprotected segments in the OR array, and Y_{1col} is the yield of a single bundle of W_{out} output signals. The unprotected OR array segments include all of the wire crossings and fanouts. They are considered to be part of the chip kill section as failures impact multiple modules. The expression is

$$Y_{mk_ORarray} = Y_{crossing}^{5W_{out}+5+3(W_{out}-1)(W_{out}-2)+3R(W_{out}-1)} \cdot Y_{fanout}^{R(5W_{out}-3)}. \quad (10.40)$$

The yield for a single cluster of W_{out} output bits combines the R configuration registers, the AND2 and OR2 gates from the enable logic, and short, long, and corner wire segments. The expression becomes

$$\begin{aligned} Y_{1col} = & Y_{register(1)}^R \cdot Y_{AND2}^{R \cdot W_{out}} \cdot Y_{OR2}^{(R-1)W_{out}} \cdot Y_{shortwire}^{R^2+R(1+W_{out})} \\ & \cdot Y_{longwire(N_{mod})}^{R \cdot W_{out}} \cdot Y_{corner}^{2R}. \end{aligned} \quad (10.41)$$

As with TMR and modular reconfiguration, yield of the supermodule combines the yield of the logic module and local input and output wiring. The expression is

$$Y_{supermod} = Y_{mod} \cdot Y_{inwires} \cdot Y_{outwires}. \quad (10.42)$$

The yield of the local input wiring is the same as modular reconfiguration, or

$$Y_{inwires} = Y_{shortwire}^{W_{in}} \cdot Y_{longwire(N_{mod})}^{W_{in}} \cdot Y_{corner}^{W_{in}}. \quad (10.43)$$

Finally, the yield expression for the local output wires is

$$Y_{outwires} = Y_{shortwire}^{3 \cdot W_{out}} \cdot Y_{fanout}^{2 \cdot W_{out}} \cdot Y_{corner}^{W_{out}} \cdot Y_{crossing}^{(W_{out}-1)(W_{out}-2)}. \quad (10.44)$$

10.7 Performance Comparison

This section compares the performance of the QCA fault tolerance expressions to each other, as well as to the original CMOS models. Unlike the CMOS models, QCA yield depends heavily on the physical layout and associated wire lengths. The impact of QCA cell density is shown, as well as the negative effect of silicon clock scaling. Finally, rules are proposed for fault tolerant QCA design.

10.7.1 Setup. These results are based on the analytical expressions developed in the previous sections. Defect clustering is not considered, but can be determined using the same compounding techniques used in the CMOS models. Instead, the focus of this section is the relative performance of the three QCA fault tolerance designs: TMR, modular reconfiguration, and TMR-protected reconfiguration, and their CMOS equivalents.

Two wire models are examined. In the short wire model, wire lengths are the average wire length as defined earlier. In the long wire model, short wires are used for local interconnect, while long wires are used for intermodule connections. Wire length depends on N_{mod} , θ_{qca} and $k_{silicon}$.

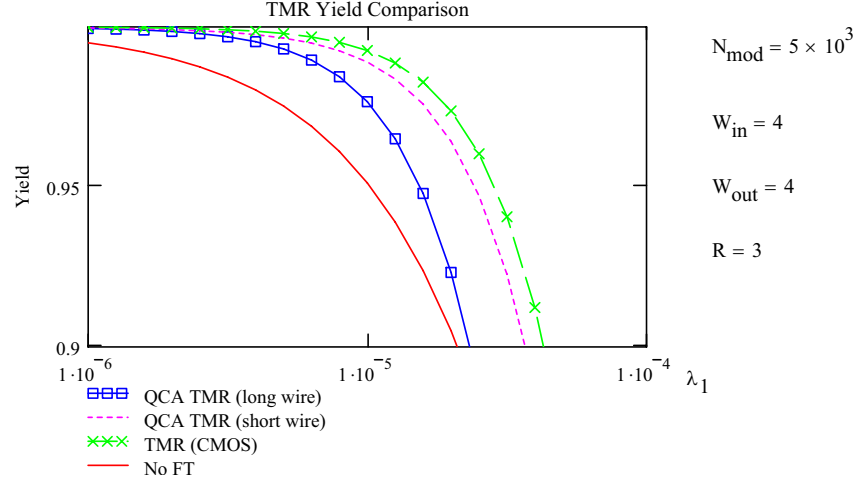


Figure 10.12: Unclustered yield for TMR using the CMOS, long-wire QCA, and short-wire QCA models. Performance of the short wire model approximates the CMOS model.

10.7.2 Observations.

10.7.2.1 QCA TMR yield. Figure 10.12 shows the unclustered yield for the three models of TMR. The module consists of 5000 devices (either QCA cells or transistors), with four inputs and four outputs. From the figure it is evident that all three models show improvement over the non fault tolerant module, with the CMOS model producing the highest yield. The short wire QCA model produces results that approach CMOS, while the long wire model predicts a lower yield. This is because the number of wire crossings and other cells in the unprotected areas is large relative to the overall number of devices in the structure, decreasing yield.

As the size of the modules increases, wiring overhead becomes small, and yield is dominated by the devices in the modules themselves. Figure 10.13 shows the maximum allowed defect probability (MADP) for 90% yield. In this case, as module size increases both QCA wire models approach the performance of the CMOS model. From the figure, the “break even” module size can be identified where the use of TMR provides an improved yield over the unprotected module. For CMOS, the break even point is $N_{mod} = 76$. For the long wire QCA model, the break even point

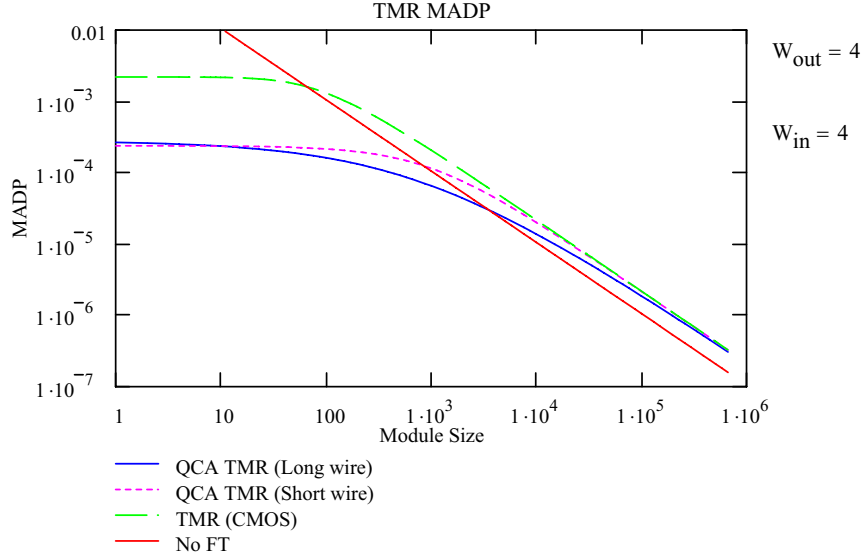


Figure 10.13: Maximum Allowed Defect Probability for QCA TMR. Performance is worse than the CMOS model for small module sizes. As $N_{mod} \rightarrow \infty$, the models converge since performance is dominated not by the wiring but by the number of devices in the modules.

is $N_{mod} = 2955$, indicating much larger module sizes are required for TMR to be effective for QCA.

10.7.2.2 Effects of Input Width. The MADP for QCA TMR with four different input widths, W_{in} , is shown in Figure 10.14. Unlike the CMOS model, QCA TMR is very sensitive to the number of bits in the input due to the susceptibility of the design to failures in the fanouts in the input lines. Figure 10.15 shows that the effect occurs in both the long wire and short wire models.

A similar effect occurs with modular reconfiguration, as shown in Figure 10.16. MADP begins to fall off significantly as $W_{in} > 10$.

The design for TMR-protected reconfiguration is not as strongly impacted by increasing W_{in} , as shown in Figure 10.17, due to the improved fanout design for the inputs.

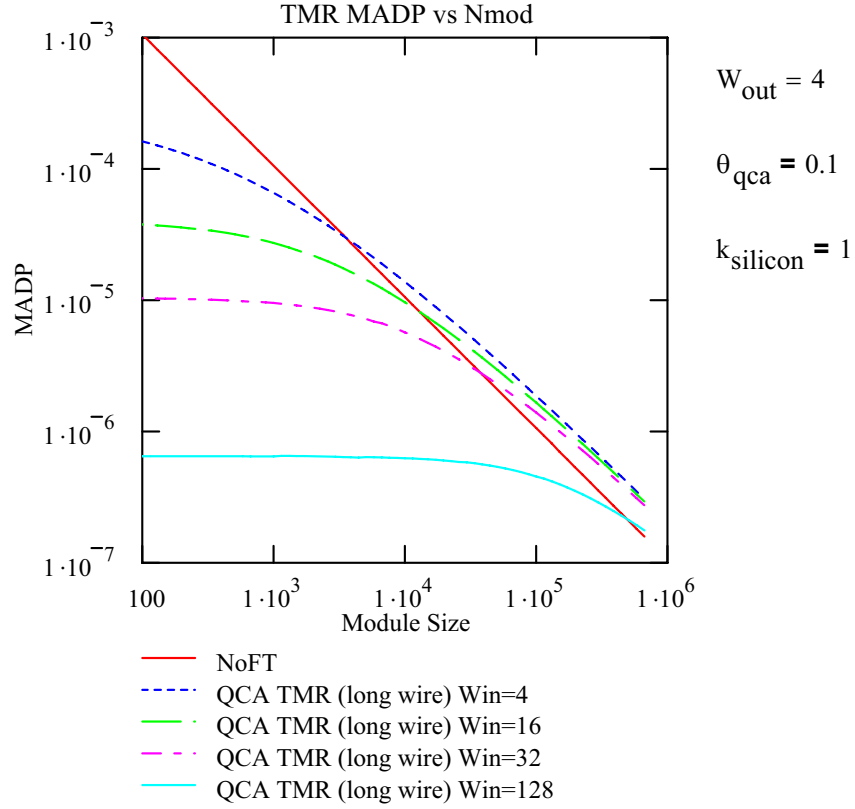


Figure 10.14: Max allowed defect probability for QCA TMR for four different input widths. Increasing W_{in} requires a larger module size for TMR to provide any benefit.

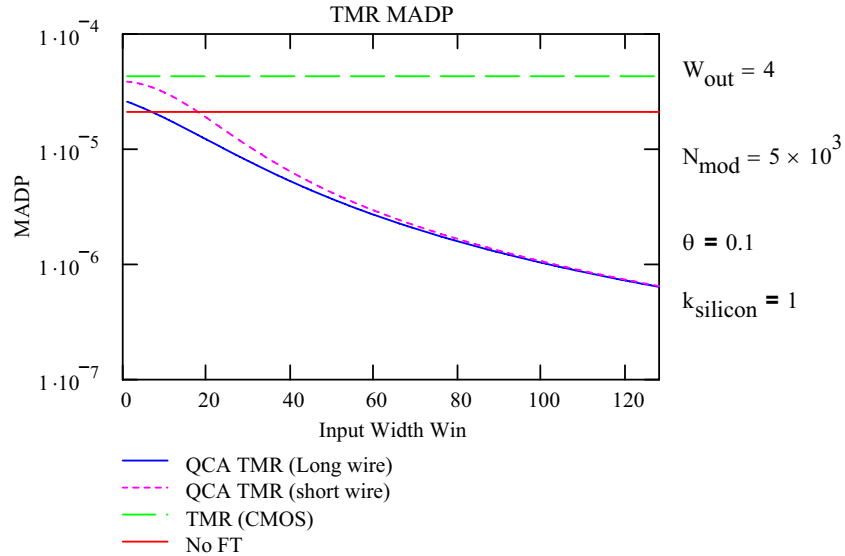


Figure 10.15: TMR MADP versus input width W_{in} . The CMOS model is independent of W_{in} . For this example, QCA TMR is ineffective for $W_{in} > 10$.

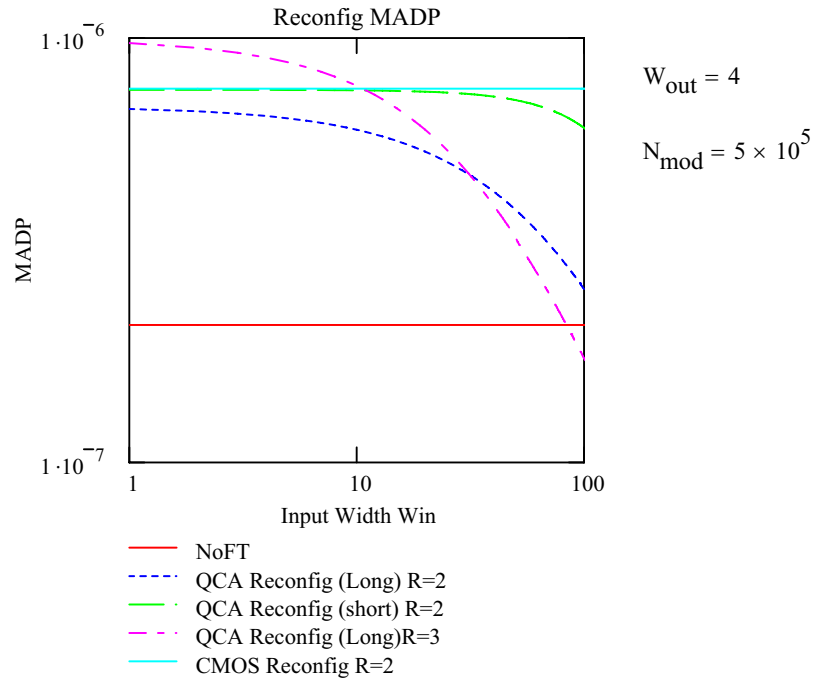


Figure 10.16: MADP versus input width W_{in} for modular reconfiguration. Reconfiguration performs less effectively as W_{in} increases. The Y axis is shown with a logarithmic scale

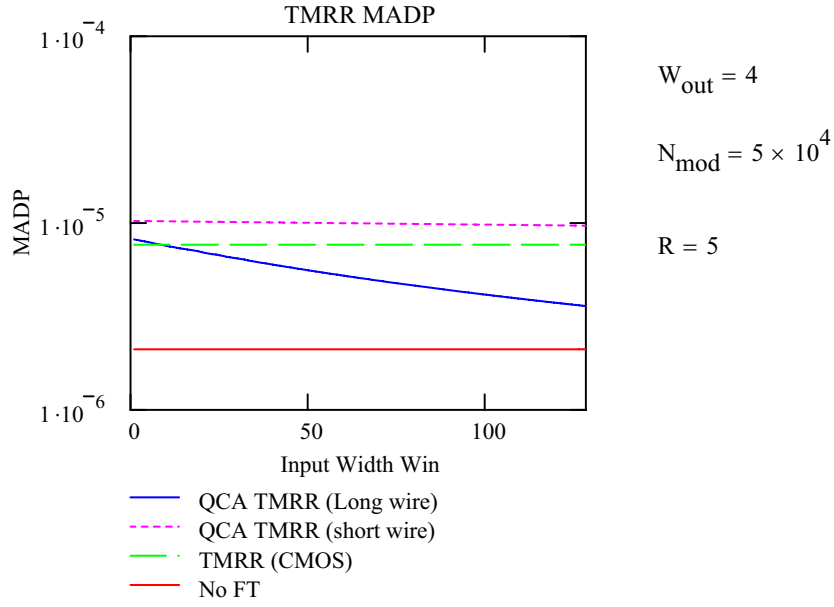


Figure 10.17: CMOS TMRR is unaffected by input width. For QCA, the long wire model is significantly affected.

MADP for the three fault tolerance designs is shown together in Figure 10.18. Both TMR and modular reconfiguration MADP falls as W_{in} increases. While the MADP of TMR-R falls as well, it is less strongly affected. Of the three designs, TMR-R is best suited for circuits with a large number of inputs.

10.7.2.3 Output Width. The MADP of the TMR models versus increasing output width W_{out} is shown in Figure 10.19. CMOS performance degrades slowly with W_{out} in this example. QCA performance degrades very quickly for both the short and long wire models.

The MADP for QCA TMR for four different output widths is shown in Figure 10.20. A similar plot for QCA modular reconfiguration is shown in Figure 10.21. From the plots, it is evident that modular reconfiguration degrades less quickly than TMR with larger output widths.

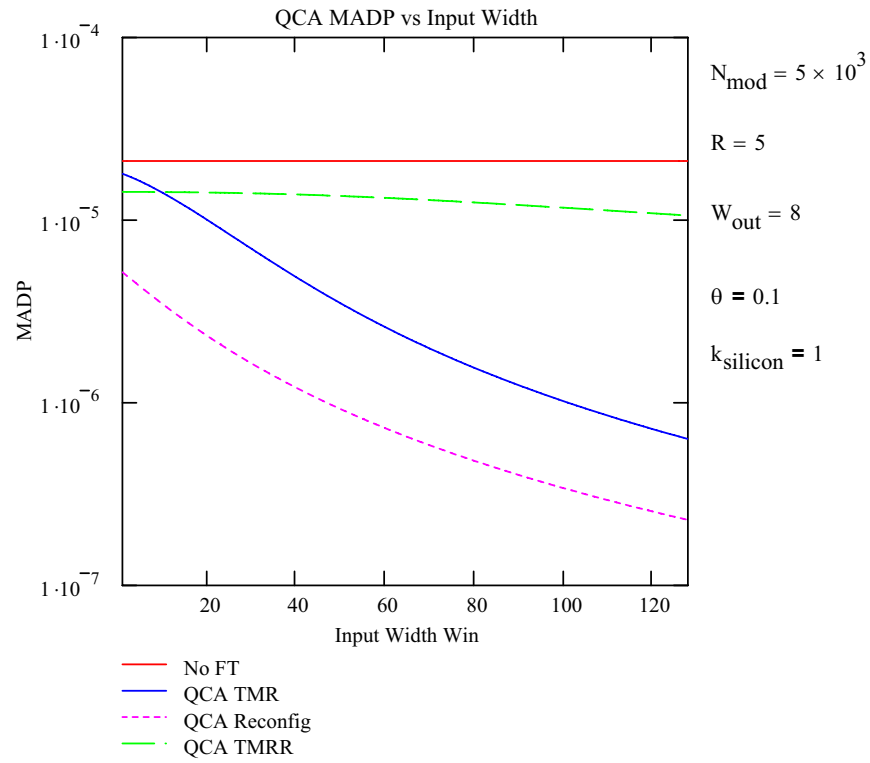


Figure 10.18: MADP versus input width W_{in} for the three fault tolerance structures. TMR-R is the least impacted by increasing input widths.

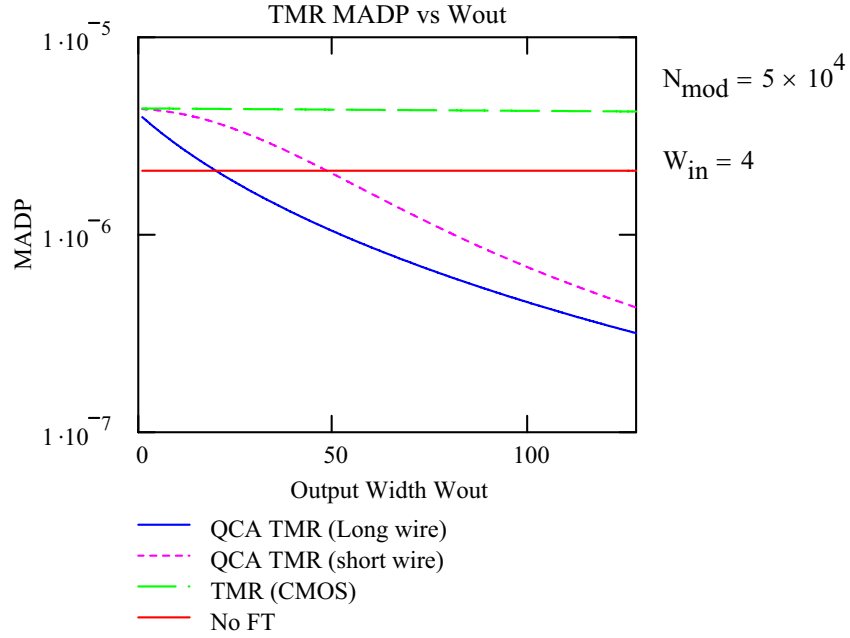


Figure 10.19: MADP versus Output Width W_{out} . All models decrease in performance as W_{out} increases. for CMOS, the decrease is much less than in QCA.

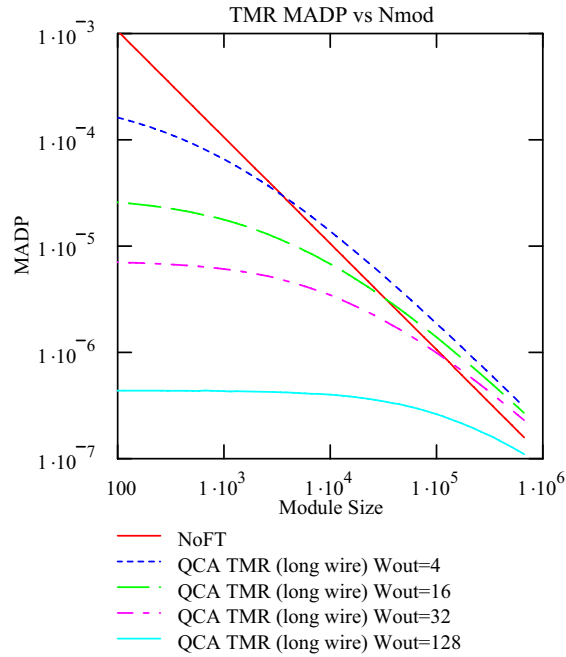


Figure 10.20: QCA TMR MADP for four values of W_{out} . Increasing W_{out} requires larger module sizes for TMR to provide a benefit.

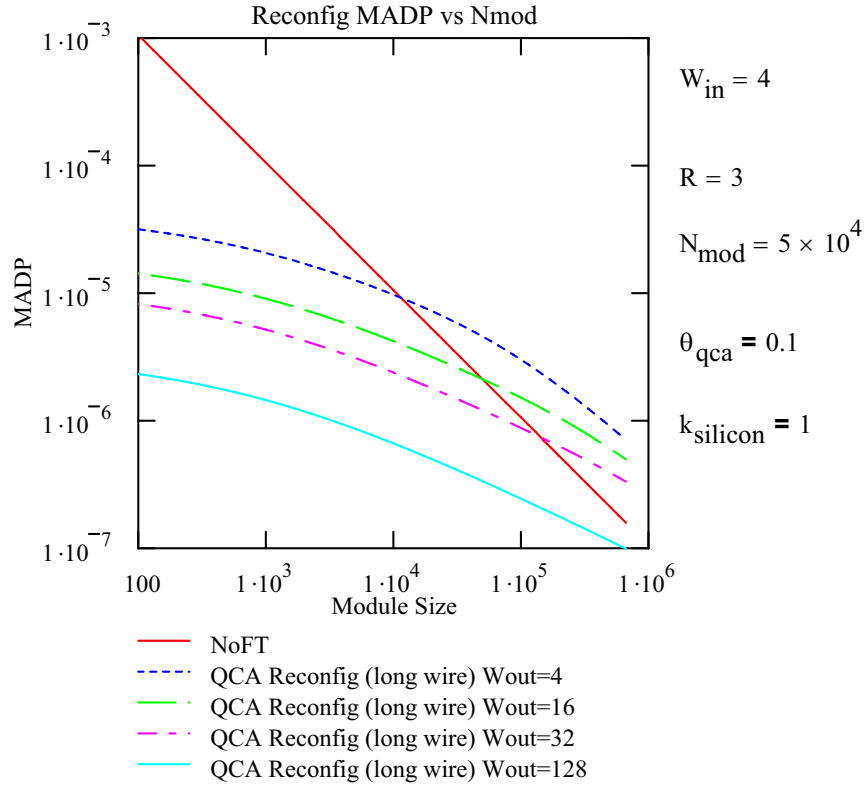


Figure 10.21: MADP of modular reconfiguration versus output width, W_{out} . Reconfiguration is less impacted by output width than TMR due to fewer wire crossings and lack of majority gates.

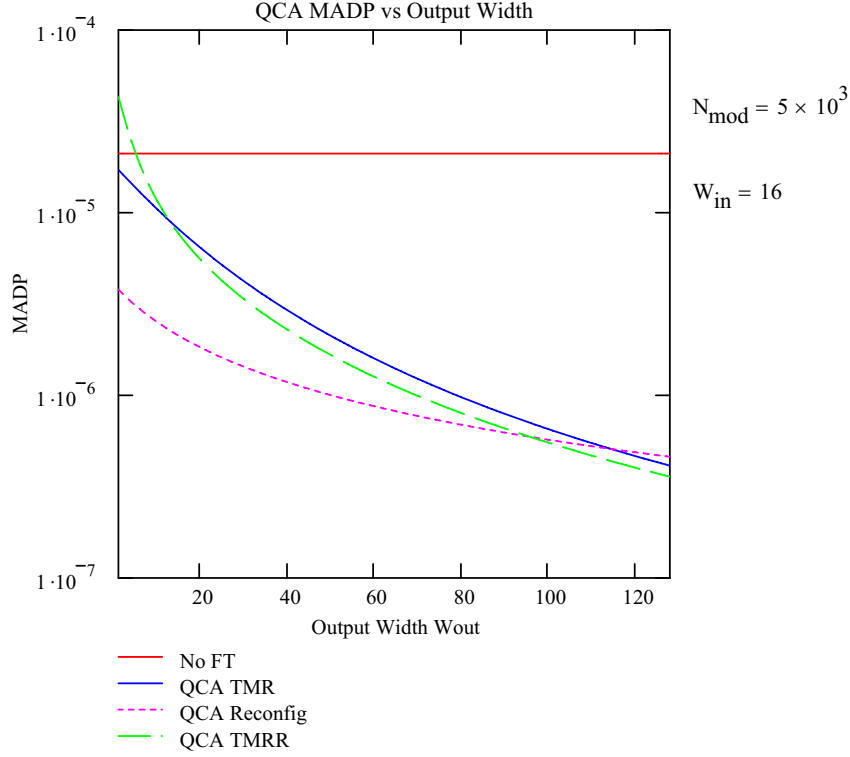


Figure 10.22: MADP versus output width W_{out} for the three fault tolerance structures. Modular reconfiguration is the least degraded by increasing the width of the outputs.

Figure 10.22 is the MADP of all three designs versus increasing W_{out} . While all degrade with increasing W_{out} , modular reconfiguration has the slowest rate of decrease, and is well suited for circuits with large numbers of outputs.

10.7.2.4 QCA Cell Density. QCA cell density is a new parameter, scaling the average length of long wires depending on the size of the module. It reflects that fact that empty spaces on the substrate are required to separate unrelated devices. The MADP for QCA TMR versus cell density, θ_{qca} is shown in Figure 10.23. Increasing the cell density for a given module size results in shorter length and width dimensions for the module, and shorter long lines. Thus, yield and MADP improve. The effect is most noticeable for small modules, since the relative change in long wire length is not as significant for large values of N_{mod} .

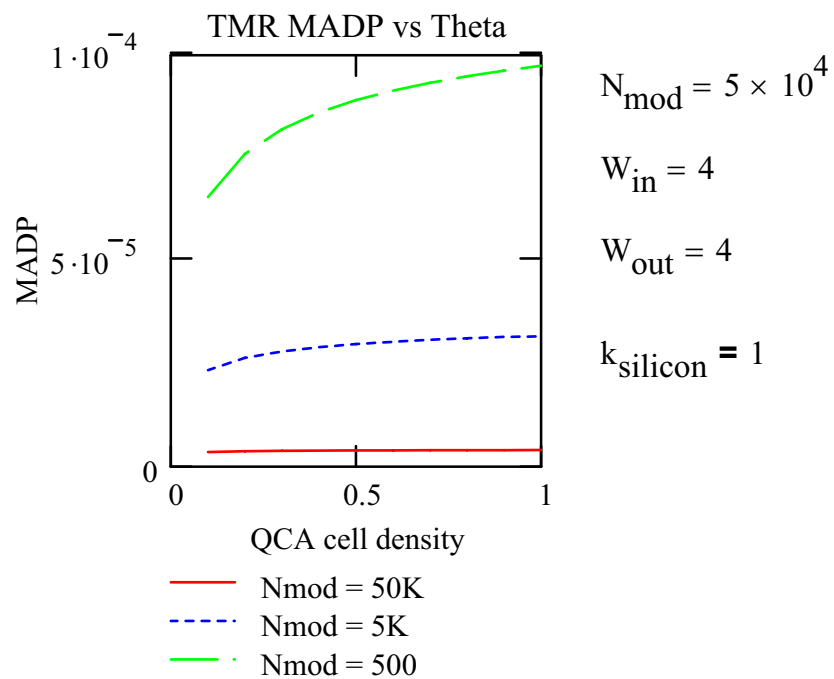


Figure 10.23: TMR MADP versus QCA cell density θ_{qca} . Increasing the density of QCA cells increases the MADP. The effect is more apparent for small modules.

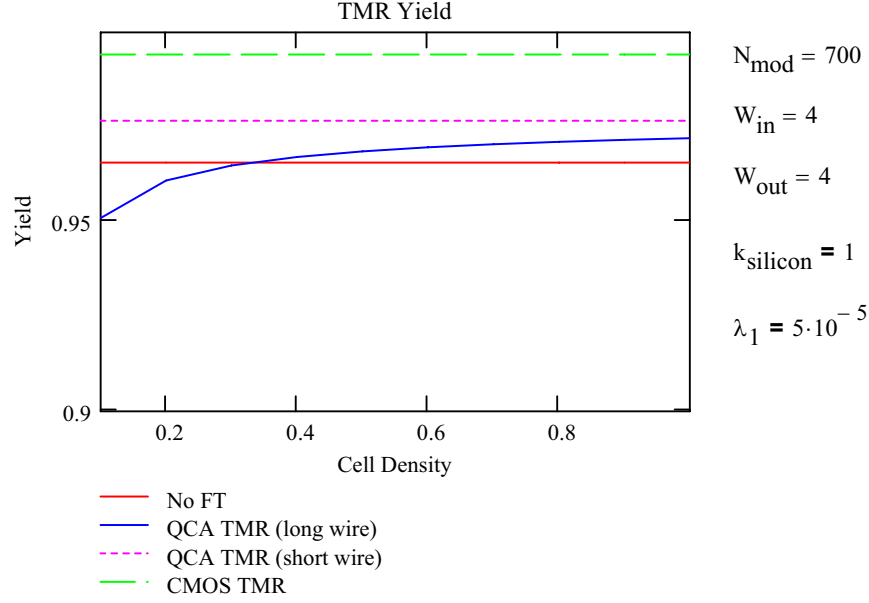


Figure 10.24: TMR Yield versus QCA cell density. The short wire model does not account for θ_{qca} and does not change. For the long wire model, density can determine whether TMR provides any benefit over a non fault tolerant design.

Figure 10.24 is an example of cell density as the determining factor in whether TMR provides an improvement over an unprotected module. For a low density layout, TMR results in a smaller yield than the unprotected design. A higher density layout increases TMR yield above an unprotected design.

Increasing cell density can have a negative effect. Separation of unrelated wires and gates decreases the probability a lateral displacement defect disrupts circuit operation. Thus, placing cells closer together ultimately increases the defect probability, λ_1 . While difficult to model, there will be a *best density* such that yield is maximized by keeping wire lengths as short as possible but λ_1 remains small.

10.7.2.5 Silicon Clock Scaling Factor. The silicon clock scaling factor, k_{silicon} , is another parameter unique to QCA. It reflects the increase in the QCA design required when the minimum feature size of the silicon clock wires determines the size of the QCA layout. Increasing the scaling factor results in longer wires throughout the QCA design, and has a negative impact on yield and MADP performance. This

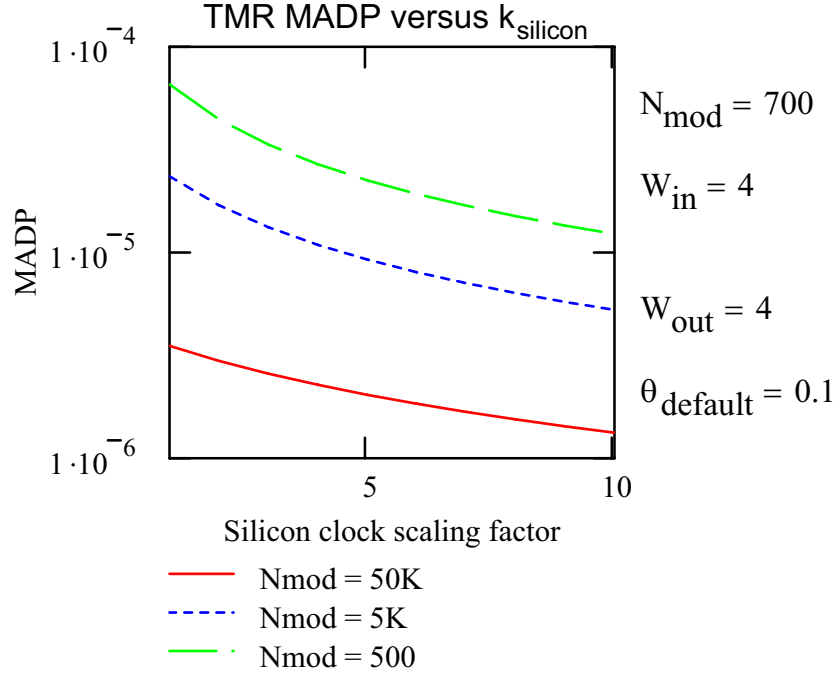


Figure 10.25: TMR MADP versus QCA silicon clock scaling factor k_{silicon} . Fault tolerance performance decreases if the silicon clock lines require QCA scaling. If $k_{\text{silicon}} > 1$, longer QCA lines are required, and yield decreases due to failures in the long lines.

effect is shown in Figures 10.25 and 10.26. The MADP for long wire QCA TMR decreases significantly as k_{silicon} increases from one to ten. The break even point versus an unprotected approach appears to shift significantly to the right, implying larger modules are required to achieve a benefit with TMR. However, the size of the unprotected module will also increase due to clock scaling, moving the straight line denoting the unprotected module down as well.

10.7.2.6 Redundancy. Both modular reconfiguration and TMR-protected reconfiguration can benefit from the addition of additional modules. In many cases, there is an upper limit on the number of redundant modules before yield begins to decrease due to the overhead in the switching hardware and wiring. For the example shown in Figure 10.27, the best MADP is achieved by $R = 5$.

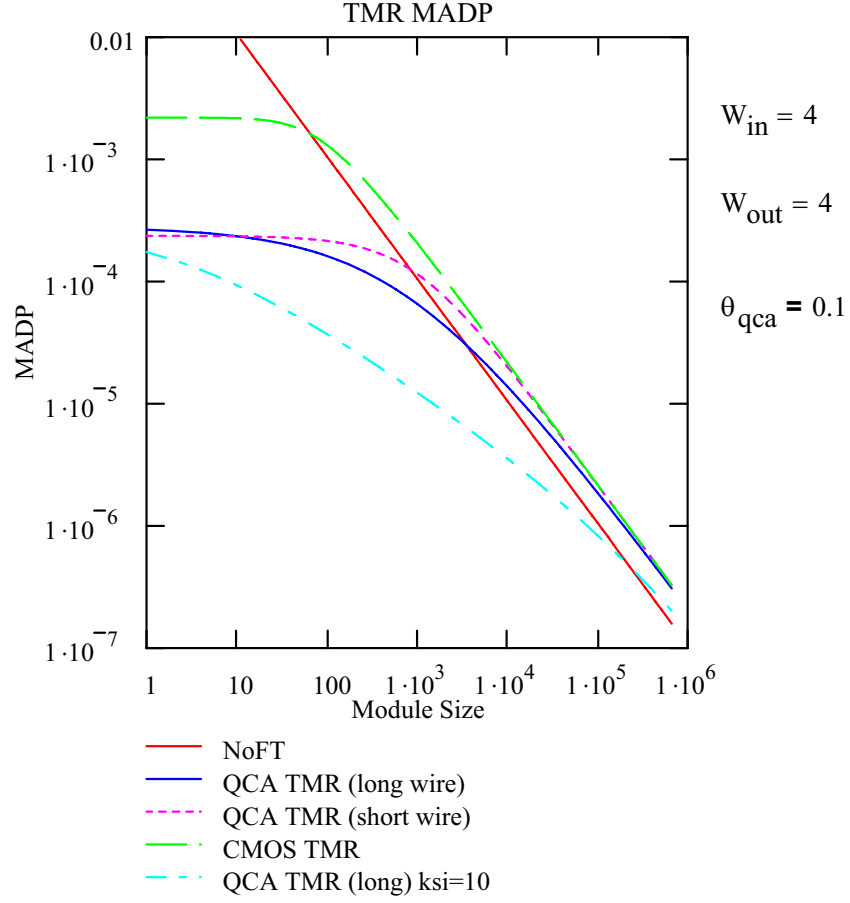


Figure 10.26: TMR MADP for both CMOS and QCA models. Two values of $k_{silicon}$ are used. If scaling is required due to the silicon clock lines, much larger module sizes are required to achieve as benefit over the non-fault tolerant approach.

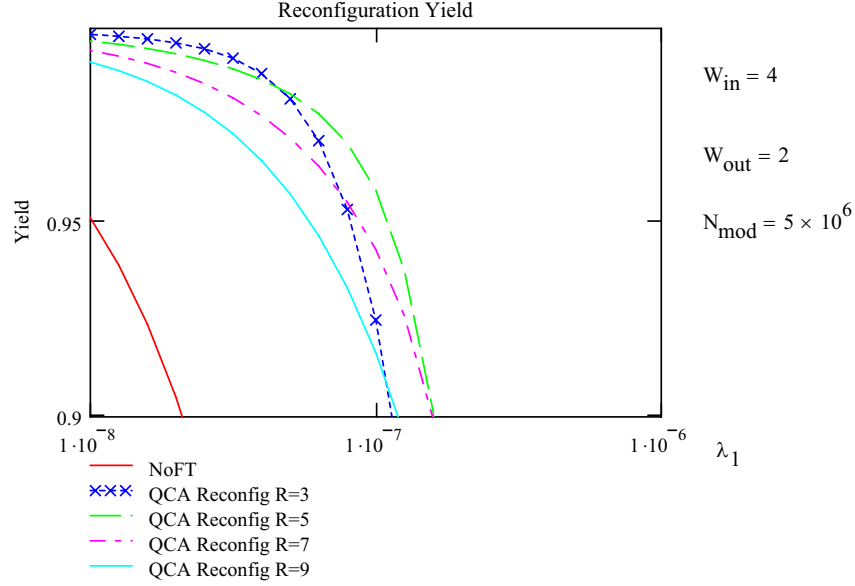


Figure 10.27: Yield of QCA modular reconfiguration versus R . For large modules, yield is improved by adding additional modules. In each case, there is a point of diminishing return, beyond which additional modules actually decreases performance.

10.7.2.7 Comparison of Techniques. The relative benefit of the three fault tolerance schemes depends on the specific configuration of module size, number of inputs, and number of outputs. In Figure 10.28, for example, modules smaller than $N_{mod} = 10,000$ get no benefit from the fault tolerance circuits. For larger modules, TMR-R provides the most benefit.

Comparison of the three QCA models against the original CMOS models shows the impact of failures in the interconnect. Figure 10.29 shows that MADP for the QCA circuits is in all cases less than the CMOS designs. Larger modules are required to achieve a yield benefit. The figure also illustrates how the best fault tolerance technique changes depending on the technology used. For this example, CMOS TMR initially provides the greatest MADP in the region $N_{mod} > 100$, but is quickly replaced by modular reconfiguration. For QCA, TMR-R provides the greatest MADP.

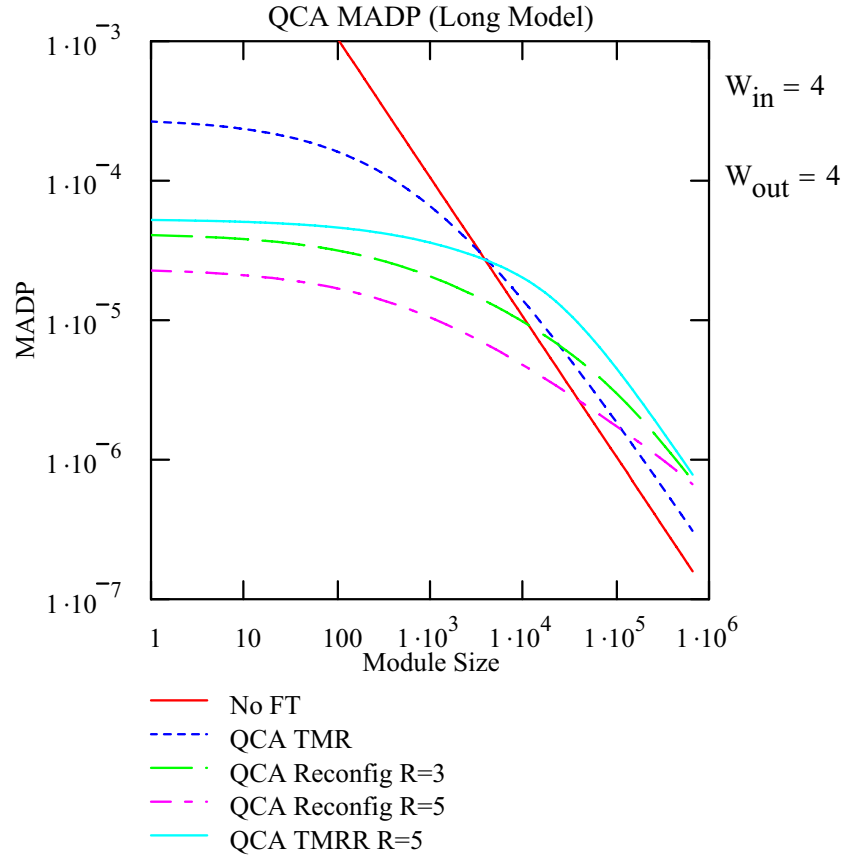


Figure 10.28: Comparison of MADP for three different QCA FT structures. Initially, TMR performs better than reconfiguration or TMRR, but performance is still worse than if no fault tolerance had been used at all. For large module sizes, TMR-R provides the best yield.

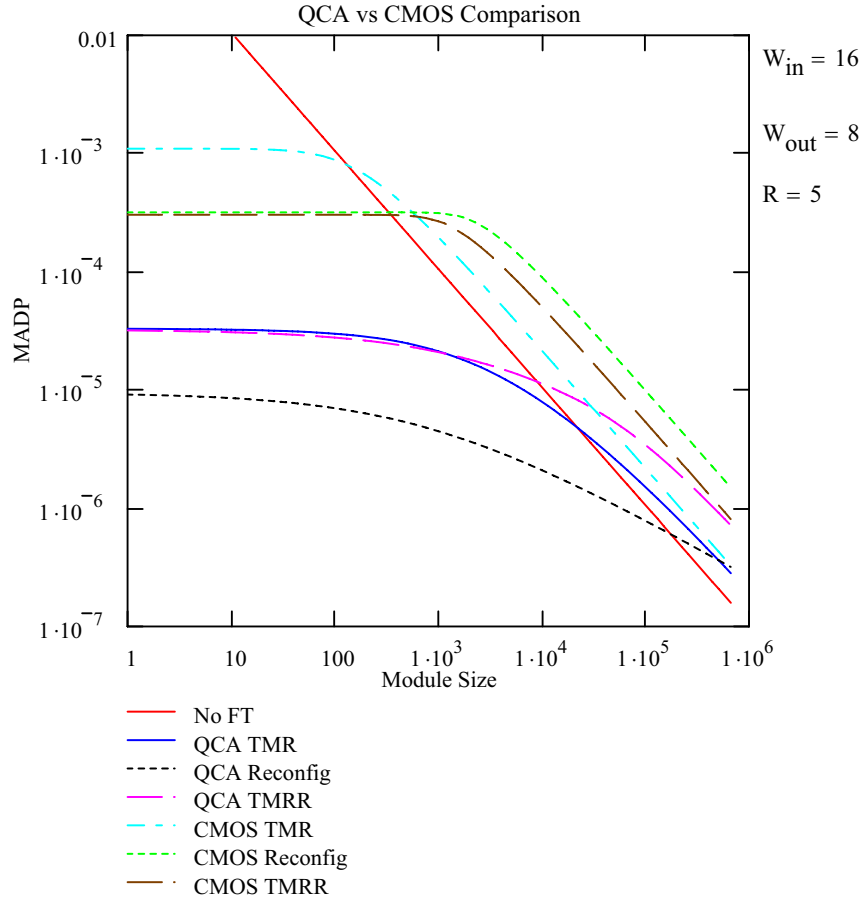


Figure 10.29: Comparison of MADP for CMOS and QCA. In all cases, QCA performance is less than CMOS, requiring larger module sizes to be effective. Sometimes the choice of the best fault tolerance approach is different for QCA than CMOS. In this example, reconfiguration provides the highest yield for CMOS for large modules, while TMR-R is the best choice for QCA. For reconfiguration and TMR-R, $R = 5$.

10.7.3 Designing for Fault Tolerance. Based on the analysis of the previous section, it is possible to draw conclusions that will be helpful for designers of fault tolerant QCA circuits.

Minimize wire lengths. While wires must sometimes be lengthened for clock synchronization purposes (i.e., to cause multiple inputs to arrive at a gate at the same time), wire length should be minimized to reduce the probability of wire faults.

Minimize wire crossings. Faults in wire crossings can impact multiple modules, causing chip kill effects. While unavoidable in a two-dimensional technology such as QCA, careful layout can help to minimize the number of crossings.

Use narrow modules facing the output bus. The models in this section assumed the modules are square. A rectangular layout with a narrow side facing the output bus would result in shorter long wires in the bus and improved reliability.

Split input lines as early as possible. This maximizes the number of cells protected in the supermodules and reduces the number in the chip kill section of the circuit prior to the fanouts.

Use large modules when possible. Larger modules generally benefit more from fault tolerance. However, modules with large numbers of inputs and outputs can eliminate the benefit. The best choices are modules with large size and few inputs and outputs. Design partitioning is very important.

Consider W_{in} and W_{out} when determining fault tolerance approach. The choice between modular reconfiguration, TMR, and TMR-R depends not only on module size, but on the number of inputs and outputs. As shown, the best choice depends on all of these factors.

Balance cell layout density θ_{qca} against increasing λ_1 . Increasing density of the layout will decrease the average wire lengths, but will increase the defect probability. A maximum density layout may not provide the best yield.

10.8 Conclusions

This chapter examines the problem of implementing the fault tolerance schemes described in previous chapters on a non-CMOS device technology. Many of the device technologies offered as a replacement for silicon CMOS are more difficult to fabricate, and suffer higher defect rates. Adaptation of fault tolerance techniques to work with these technologies is an essential step in practical adoption. Quantum cellular automata is one candidate technology. Logic structures and interconnect are formed from the same QCA cells. Thus, cost, power, and yield models must all account for the resources used in wiring.

This chapter also develops models for hardware cost, power, and speed for QCA circuits. The problem of accurately estimating wire lengths prior to actual layout is abstracted to two wire models: short wires, intended for local interconnect, and long wires for intermodule communication. Long wire lengths depend on the module size, N_{mod} , and two new parameters: θ_{qca} , the QCA cell fill density, and $k_{silicon}$, the silicon clock scaling factor. Cell fill density adjusts the wire length estimate to reflect the large number of vacant cells in the QCA substrate. The silicon clock scaling factor accounts for an increase in QCA layout size if silicon clock wires are used. In this case, minimum feature size can be determined by the lithographic process used to create the wires rather than the size of the QCA cells. Clock scaling causes a significant decrease in yield due to the large numbers of additional cells in the long wires.

Defect models for the three main fault tolerance schemes are developed. First, QCA defects are examined and shown to be directly dependent on the number of cells in the module, N_{mod} . Thus, the use of a single defect probability, λ_1 , is possible. Circuit diagrams for TMR, modular reconfiguration, and TMR-protected reconfiguration are developed for QCA. Yield expressions for these models are developed and shown to be scalable for modules of different sizes and larger numbers of inputs and outputs.

The yield performance of the QCA fault tolerance models is more dependent on layout than CMOS, as well as being dependent on additional factors such as number of inputs, cell fill density, and silicon clock scaling factor. While the fault tolerance techniques are beneficial to QCA circuits, design tradeoffs are significantly different and require careful design. For example, high cell density increases yield by shortening line lengths, while potentially decreasing yield due to increased defect rates caused by close cell placement. From this, a set of design guidelines for fault tolerant QCA circuits are proposed. The fault tolerance techniques described in previous chapters are applicable to QCA, with some modification.

XI. Conclusions and Recommendations

This research studies the problem of implementing a microprocessor using device technologies less reliable than modern silicon CMOS. While process scaling with silicon CMOS is likely to continue for the next decade or more, fundamental limits do exist. The end of silicon scaling is inevitable and other technologies must one day supplant silicon CMOS. While no device technology has yet emerged as a clear successor, all known alternatives are more difficult to manufacture than CMOS, suffering higher rates of manufacturing defects and operational failure. To be useful, means must be found to implement complex architectures with acceptable yield. Given the large numbers of devices that will be available in the future, using a portion to provide fault tolerance at the architectural level is possible.

11.1 *Conclusions*

The focus of this work is primarily at the architectural level. An architecture for a fault and defect tolerant microprocessor was created and demonstrated to be realizable using technologies much less reliable than silicon. Mathematical models were developed for the processor and supporting fault tolerance techniques. Using these techniques, the detailed functional architecture was developed.

11.1.1 Goal Status. The goals of this research, as described in Chapter III, have been achieved.

To achieve goal one, a system architecture for the fault and defect tolerant microprocessor was proposed. A concept of operations, addressing yield testing, startup configuration, operational testing, and fault detection and recovery was proposed. Required functions were identified, and a means for comparing a fault and defect tolerant architecture with conventional designs were examined.

Using Monte Carlo simulation, manufacturing yields of 70% are achievable with device defect probabilities as high as 10^{-6} , fully three orders of magnitude greater than modern CMOS-based designs. Mathematical techniques to estimate yield were

adapted from modern yield modelling techniques. Several of the limitations of these techniques were discussed in Chapters V and X.

To achieve goal two, a functional architecture was developed. The cache memory was the most critical part of the processor, and an effective architecture was proposed to provide acceptable performance. Acceptable yields can be achieved for both the cache memory and the overall processor.

For goal three, the primary fault tolerance techniques required by the FDT processor were analyzed using QCA as the underlying technology. The techniques can be used directly with QCA, although the benefits can be reduced or increased due to unique characteristics of the device technology. Mathematical models for yield and hardware cost were developed for QCA implementations of TMR, modular reconfiguration, and TMR-R. Using these, it is possible to develop yield models for an entire processor implemented in QCA. However, since the results depend greatly on wiring lengths, more accurate estimators for wire length may be required.

Finally, goal four improves the mathematical models for classic fault tolerance schemes. Chapter VI developed the first accurate model for von Neumann multiplexing. The technique has a great deal of potential for use in device technologies that are much smaller than silicon CMOS, but much less reliable. Accurate estimation of the fault tolerance performance of this technique at moderate levels of redundancy is now possible.

11.1.2 The Big Picture. Further work must be done to answer the biggest question associated with this work: “How reliable must a device technology be to implement a microprocessor?” It was proven that acceptable yields can be achieved using devices with defect rates exceeding 10^{-6} , using only 15 times the number of devices required for the baseline design. Chapter VI demonstrated that high yields are achievable with defect rates as high as 10^{-2} using aggressive fault tolerance techniques such as von Neumann multiplexing (or other methods such as fine-grain reconfiguration). The cost of fault tolerance in this range is high, as redundancy factors of

100 times or more become necessary. Depending on the size improvement of a future device technology relative to silicon CMOS, this penalty may or may not be acceptable.

11.2 Contributions

Research contributions include:

1. The first accurate analytical model for von Neumann multiplexing [RBB06].
2. A new fault tolerance technique, TMR-protected reconfiguration, that combines the benefits of TMR and modular reconfiguration.
3. A defect tolerant CAM-based cache architecture [RBMK06b, RBMK06a].
 - Proposed the first known use of extended Golay code in fault tolerant cache design.
 - Created a novel bus replication technique to reduce chip kill effects, enabling the use of a large CAM.
 - Unlike most work in memory fault tolerance, analyzed the entire cache architecture.
 - Demonstrated that 90% yields could be obtained with defect probabilities greater than 10^{-6} , three orders of magnitude better than conventional designs.
4. A fault and defect tolerant processor architecture.
 - Proposed a concept of operations and identified required capabilities.
 - Developed an analytical model for the yield of the FDT processor.
 - Demonstrated that 70% yield could be obtained with defect probabilities greater than 10^{-6} .
5. Demonstration of how fault and defect tolerance techniques can be applied to QCA, a non-silicon based device technology.

- Created QCA implementations of TMR, modular reconfiguration, and TMR-R.
- Derived analytical expressions for the yield performance of these techniques with QCA.

11.3 Recommendations for Future Research

The following recommendations are made for further research:

- Develop mathematical models for soft error performance and operational reliability. As discussed in Chapter II, soft errors and SEUs are becoming more common. While the FDT architecture incorporates protections at the hardware level, the benefit was not quantified, and further work should be done to determine whether operating requirements can be met.
- Investigate operating system and application level support for the fault and defect tolerance techniques proposed in Chapter VII. New instructions and operating system routines are required to support BIST/R functions. These functions will add complexity to the processor design. The effect on reliability and application performance should be quantified as another aspect of the comparison with silicon CMOS.
- Investigate fine-grained reconfigurable architectures for use in FDT architectures. FPGA-like architectures have been proposed for defect tolerant computing, offering large numbers of reconfigurable devices at a considerable cost in overhead. In addition, significant challenges exist in online testing and runtime partial reconfiguration of FPGAs. Solutions must be found before an FPGA-like architecture can be used as a FDT processor.
- Investigate the impact of fault and defect tolerance on software performance. To replace silicon CMOS, the new device technology must provide a performance benefit. While the new devices will be smaller and faster than silicon CMOS transistors, the fault tolerance overhead at the circuit and architectural levels

may negate application level speedup. Given the tremendous cost involved in switching technologies, the benefits of the new technology must be quantified before an investment is made.

- Develop models for yield and hardware cost for other device technologies. Besides QCA, other device technologies have been proposed as successors to silicon. While device-level research continues in each of these technologies, fault tolerance research must also be done to determine overall suitability for use in large circuits.

This research was directed primarily at the problem of manufacturing yield for the DFT microprocessor. While soft error tolerance was addressed and included in several aspects of the architecture, mathematical models should be developed to quantify the benefits of these techniques. From this analysis, it can be determined if the proposed techniques are sufficient for long term continuous operation.

Several of the techniques proposed in Chapter VII require the support of the instruction set and operating system. Instructions sets and computer architectures must be developed in conjunction with the operating system. Specifically, BIST/R functions should be added to both the instruction set and the operating system.

Fine-grained reconfiguration, similar to that of modern FPGAs, have the potential for much higher reliability than the medium-grained techniques used in this architecture. However, fine-grained reconfiguration requires much higher levels of redundancy. In addition, significant challenges must be overcome to use reconfiguration, including dynamic routing, support for partial reconfiguration, and protection from soft errors. Potential research goals in this area are included in Appendix B.

The fault tolerance methods used in the FDT processor introduce performance overhead. Encoders and decoders used for ECC introduce direct delays on pipeline performance, requiring slower clock speeds. Other architectural decisions, such as the use of a CAM-based cache, introduce further penalties. In addition to high yield, the

FDT processor must compete favorably with silicon CMOS in terms of application performance. The impact of fault tolerance schemes should be studied as well.

Finally, further research should be done using specific device technologies. Chapter X showed how fault tolerance performance can vary significantly depending on the specific capabilities and limitations of the device technology. Consideration of these factors is essential to determine whether a device technology should be adopted.

Appendix A. Additional Background

This appendix provides additional background information on programmable logic devices and reconfigurable computing. Programmable logic devices, including FPGAs, are one method of implementing reconfiguration. As described in Chapter II, reconfiguration is a very effective method of fault tolerance. Modern FPGAs typically allow reconfiguration at a finer level of granularity than that used in the modular reconfiguration approach in this document. Thus, there is some interest in using an FPGA-like architecture for fault and defect tolerant architectures. This appendix provides background on these devices.

- Section A.1 discusses Programmable Logic Devices (PLD); these circuits can be configured to perform arbitrary logic functions. A key fault tolerance technique, called reconfiguration, requires capabilities such as those provided by PLDs. The regular, configurable structure of the PLD may be a key element in fault and defect tolerant (FDT) computer system designs.
- Section A.2 introduces Reconfigurable Computing. Reconfigurable computing systems use programmable logic devices to provide applications customized hardware with better performance than that obtained by general purpose microprocessors. Many of the techniques developed for reconfigurable computing will also be useful in defect tolerant computing.

A.1 Programmable Logic Devices

Programmable logic devices (PLD) are digital integrated circuits that contain structures whose logical function can be set or programmed after device manufacture. Early devices, such as programmable logic arrays (PLA), were fuse-based and once programmed, could not be modified further. Newer devices, such as SRAM-based Field Programmable Gate Arrays, can be reprogrammed “in the field” to change their function any number of times. Early PLDs were limited in the number of devices that could be implemented and thus were typically limited to small scale applications and

use as “glue logic” to implement connections between larger, fixed-design Application Specific Integrated Circuits (ASIC). As more advanced CMOS processes provided more devices on a chip, FPGAs became more complicated, allowing larger designs to be implemented. Current FPGAs can implement the equivalent of more than one million boolean logic gates, and are being used to implement extremely complex designs previously limited to ASICs.

PLDs have gained widespread use in electronics. The cost of an FPGA design is typically much lower than that of an ASIC, and the design time is shorter. ASIC designs can require months to fabricate after the initial digital design is created. If design errors are found, the process must be repeated. One of the most common uses for FPGAs is for *Rapid Prototyping*. A prototype digital design can be implemented on an FPGA in hours, allowing iterations of the design to be tested and refined extremely rapidly. In the past, the final design was converted to a VLSI layout for implementation as an ASIC. Increasingly, the final design is left as an FPGA implementation.

Programmable logic devices have, by design, very regular structures. Most PLDs are composed of two-dimensional arrays of *logic blocks* whose function can be programmed during configuration. Between the logic blocks, configurable interconnect structures allow logic blocks to be connected as appropriate to implement whatever design the user requires (cf., Figure A.1). Over the years, many alternative structures have been proposed with block sizes of varying *granularity* from individual transistors (e.g., Field Programmable Transistor Arrays [KZJS00]) up to entire processors.

The regular, configurable structure of PLDs is very similar to the likely architecture of the nanoscale devices that may one day replace silicon CMOS (cf., Section 2.3.1). The fine-grained array structure of the molecular crossbar devices discussed in Section 2.3.1.1 are very similar to Programmable Logic Arrays (PLA), shown in Figure A.2.

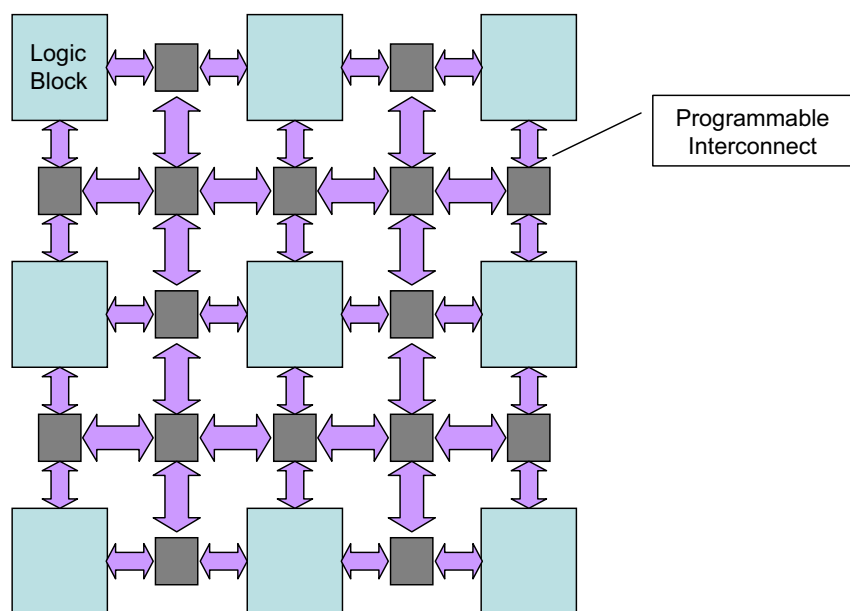


Figure A.1: A Generic FPGA Structure, composed of a two-dimensional array of programmable logic blocks, connected by a mesh of programmable interconnect lines and switchboxes.

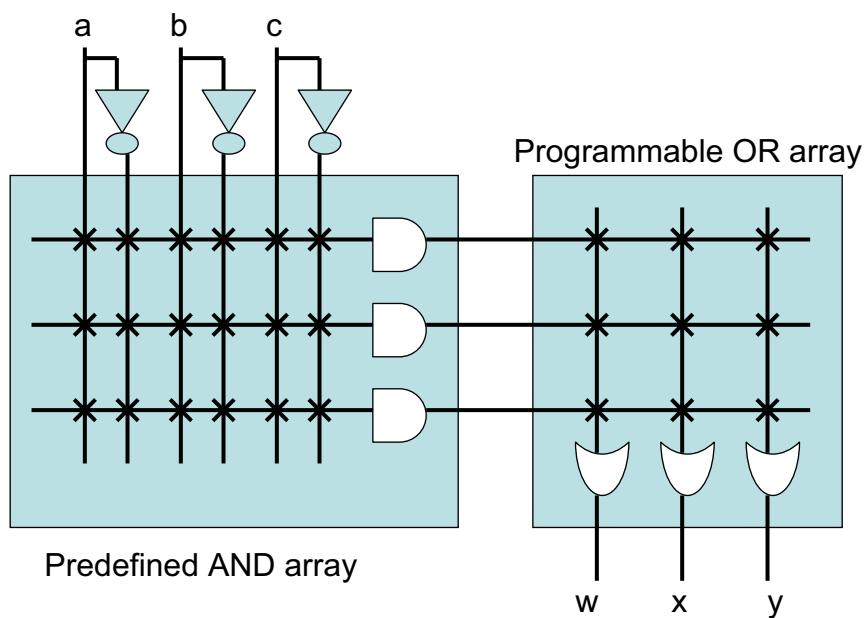


Figure A.2: Programmable Logic Array [MC04], in many ways very similar to the molecular crossbars discussed in Section 2.3.1.1.

FPGAs have been widely used in spacecraft and other applications demanding high reliability. In addition to the soft errors that change logic and memory values in ASICs, FPGAs have the additional problem of errors in the configuration memories that control the operation of the FPGA. Single Event Upsets (SEU) in an ASIC can change values stored in memory but are typically limited in scope to a single operation (i.e., the results of an ‘add’ operation may be corrupted, but the adder continues to function correctly). With an FPGA, a SEU in the configuration memory can change the operation of the adder entirely, rendering it useless. In addition, the percentage of die area devoted to registers (e.g., for configuration bits) is typically much higher in an FPGA than an ASIC. Thus, FPGAs are more sensitive to soft errors and SEUs than ASICs. Research has recently been directed to developing FPGA architectures more resistant to soft errors.

For this reason, the regular structures used in PLDs provide a good paradigm for the development of fault and defect tolerant architectures. For the nanoscale technologies that may potentially replace silicon CMOS, it will be difficult to precisely control the fabrication process. Self-assembly techniques have been proposed to create very dense, regular array structures that can be configured to perform application logic, just as with a PLD.

This section introduces programmable logic devices and their use in fault tolerant applications.

- Section A.1.1 is a general introduction to the families of programmable logic.
- Section A.1.2 discussed FPGAs in detail, including their structure, design process, and limitations. SEU effects on FPGAs are discussed, as well as several FPGA architectures intended to better tolerate these errors.
- Section A.1.3 examines runtime reconfiguration of FPGAs. Modern FPGAs can be partially reconfigured while the remainder of the device remains in operation. This capability is useful in larger scale fault tolerant architectures.

- Section A.1.4 discusses the problem of routing signals in a FPGA. The routing problem is very complex and FPGA routers can require several hours to develop the interconnect mapping for a design. Runtime reconfiguration is a desirable capability for a fault tolerant architecture. Research efforts into *dynamic* or *run-time routing* are in their infancy, but may ultimately provide a useful capability for FDT architectures.

A.1.1 Families of Programmable Logic Devices. PLDs are devices whose internal structure is determined by the manufacturer, but can be configured by the end user to perform different functions. PLDs are typically simpler and smaller than FPGAs, and the functions that can be implemented much smaller. PLDs are often categorized according to their size and complexity of the design, leading to both Simple PLDs (SPLD) and Complex PLDs (CPLD).

FPGAs occupy the middle ground between PLDs and ASICs. They are very much like PLDs but are much larger, and can implement over a million logic gates.

The function of PLDs is typically set using fuses or *antifuses*. Often the changes are irreversible, in which case the devices are called *One-Time Programmable* (OTP). Memories constructed of one-time programmable fuses are sometimes called Programmable Read-Only Memories (PROM). *Antifuses* have a high resistance in their unprogrammed state (effectively an open-circuit connection). When a programming voltage is applied, the antifuse switches to a low-resistance state, making an electrical connection [MC04].

In addition to fuses and antifuses, programmable connections can be made with Erasable Programmable Read Only Memory (EPROM) devices such as the floating gate transistor, first introduced by Intel in 1971. These devices operate by storing an electric charge on a floating gate between the normal MOSFET gate and the channel. By applying a higher than normal voltage to the control gate, an electric charge can be induced on the floating gate. When this programming voltage is removed the charge on the floating gate remains and affects the operation of the transistor, acting

as a memory. The EPROM cell is erased by applying ultraviolet light, which provides the energy necessary for the electrons trapped in the floating gate to tunnel through the gate oxide and back into the substrate.

The next improvement was the Electrically-Erasable Programmable Read Only Memory (EEPROM). These devices are similar to EPROMs, but typically have a thinner gate oxide through which electrons can tunnel at lower energies. A second, standard MOSFET electrically erases and reprograms the cell. A slightly more advanced form of the EEPROM makes up modern Flash memory .

Finally, device configuration can be stored in Static Random Access Memory (SRAM) or Dynamic Random Access Memory (DRAM). Both DRAM and SRAM require power to maintain state, and thus lose their contents when power is interrupted. DRAM is not typically used in programmable devices, as the periodic recharging of the memory transistors would be complex to control. Most modern FPGAs use SRAM to store their configuration. SRAM is easily programmed, but has the disadvantage of requiring a larger area. An SRAM memory cell (e.g., a flip-flop) requires six or seven transistors, compared to the single transistor required by an EPROM. The area required for configuration memories of a modern FPGA takes up a significant portion of the overall die and limits the size of the device. A key advantage of molecular cross-bars (cf., Section 2.3.1.1) is that configuration memory is formed by the molecular material between two perpendicular interconnect lines. The configuration memory for a programmable interconnect junction requires no additional area beyond the area of the junction itself. The hypothetical molecular crossbar FPGA is likely to be much more area-efficient than a modern CMOS FPGA.

Another alternative to CMOS SRAM configuration memory is *magnetic RAM* (MRAM). MRAM is based on the magnetic tunnel junction, first developed by IBM in 1974. MRAM has the potential to combine the high speed of SRAM, the storage density of DRAM, and the nonvolatility of FLASH. MRAM memory devices have been demonstrated, and are predicted to enter production in 2005 or 2006 [MC04].

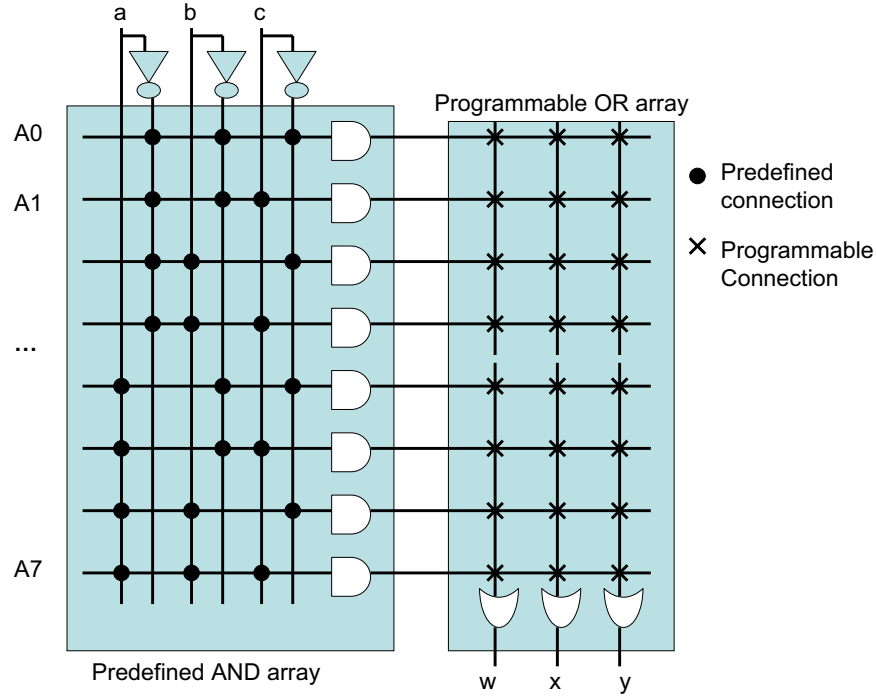


Figure A.3: Programmable Read Only Memory [MC04], composed of an AND array with predefined connections, and an OR array with configurable connections.

MRAM-based FPGAs are likely to become available before 2010, reducing several of the limitations of current FPGAs.

Simple PLDs date back to 1970 with the introduction of the PROM. PROMs consist of a predefined AND array, with a programmable OR array (cf., Figure A.3). The programmable links can be implemented with fuses, antifuses, EPROMs, or EEPROMs. Improvements to the PROM design resulted in Programmable Logic Arrays (PLA) with programmable connections in both the AND and OR arrays, and the Programmable Array Logic (PAL) which combines a programmable AND array with a predefined OR array. PLAs can be used to implement the widest range of logical functions but are slower than PLAs and PROMs.

As the number of devices that can be implemented on a chip increase, PLDs became larger. *Complex Programmable Logic Devices* (CPLD) have now replaced SPLDs in widescale use. Architectures vary between manufacturers, but most CPLD

designs combine a number of SPLD blocks together on one chip. Blocks are connected by programmable interconnect and in this sense, CPLDs are the immediate ancestor of the modern FPGA; the difference being that logic blocks in a FPGA are typically formed from SRAM-based lookup tables rather than programmable logic arrays.

A.1.2 Field Programmable Gate Arrays. The first field programmable gate arrays were developed in the early 1980s to address the gap between PLDs and ASICs. PLDs could be configured by the user, but could not support large or complex functions. ASICs can implement extremely large functions, but cannot be modified after initial fabrication. Modern FPGAs use SRAM to store their configurations, and can be easily reconfigured. They are now the most common type of programmable logic device in production. The reliability problems incurred as silicon CMOS process size shrinks will impact FPGA users, and thus FPGAs present an immediate application for fault and defect tolerant computing research.

A.1.2.1 FPGA Structure. The structure of the FPGA is very dense and highly regular. It consists of a two-dimensional array of logic blocks within a mesh of programmable interconnects. The functions of each logic block can be configured, as well as the connections between the logic blocks and the interconnect structure. In this way, arbitrary logic functions can be implemented in a FPGA. A notional architecture is shown in Figure A.4.

The internal structure of a Configurable Logic Block (CLB) can vary. Figure A.5(b) shows the most common CLB structure of one or more RAM-based *lookup tables* (LUT). Figure A.5(a) shows another alternative which provides a collection of logical gates, multiplexers, and flip flops that can be connected in various ways to implement logical functions. Complex devices can be incorporated into CLBs such as adders, multipliers, or even entire processors. The amount of logic contained in the CLB is known as the *granularity* of the FPGA.

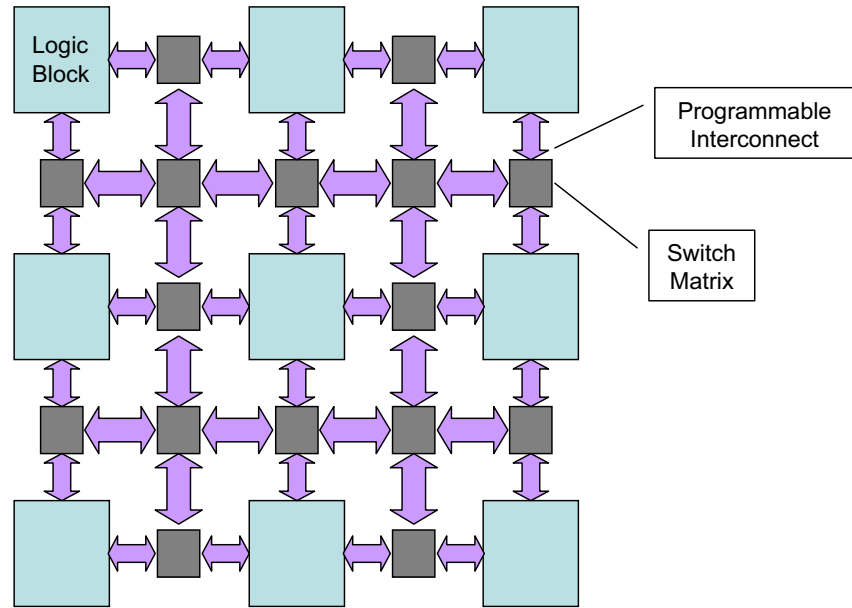


Figure A.4: A Generic FPGA Structure, composed of a two-dimensional array of programmable logic blocks, connected by a mesh of programmable interconnect lines.

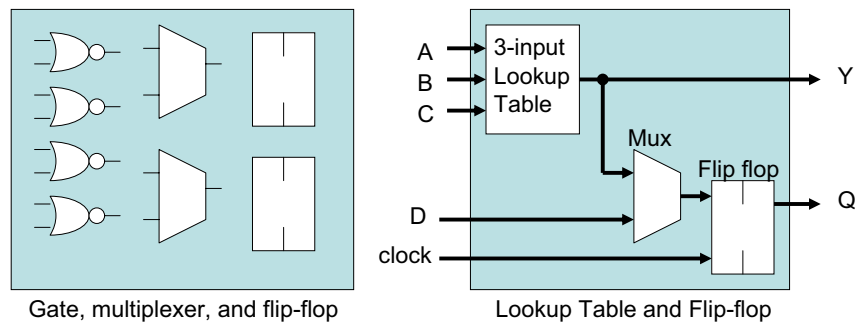
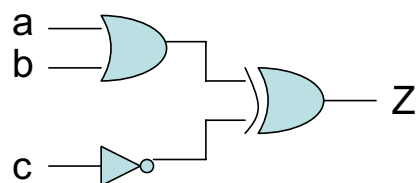


Figure A.5: FPGA Configurable Logic Blocks can be built from different devices, depending on the desired level of granularity [MC04].



Gate level design

a	b	c	Z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Three-input Lookup Table

Figure A.6: Mapping of a Circuit to a FPGA lookup table [MC04]. In this manner, any three-input, one output combinational function can be implemented.

SRAM-based lookup tables can implement any combinational logic function. The truth table for a function is simply programmed into the lookup table. In Figure A.6, a simple three input logical function is mapped to a three-input LUT. Configurable Logic Blocks are commonly composed of one or more lookup tables combined with flip-flops to provide additional memory or storage, as shown in Figure A.7.

In addition to implementing combinational logic functions, the lookup tables in the CLB can be used as memories. Figure A.8 shows three uses of a lookup table: to implement a combinational function, as a 16-bit shift register, and as a 16x1 RAM. Designers can construct larger memories by linking the LUTs in multiple CLBs.

FPGA manufacturers have attempted to find the optimum granularity for the CLB. Ideally, a FPGA would be constructed from an array of individual transistors, each connectable to form arbitrary structures. Unfortunately, the number of interconnect lines that would be required to do this render this approach infeasible and so a compromise must be made that provides adequate flexibility to the designer while minimizing the overhead of the interconnects. The three or four-input lookup table has been adopted by most FPGA manufacturers. The Xilinx FPGA shown in Figure

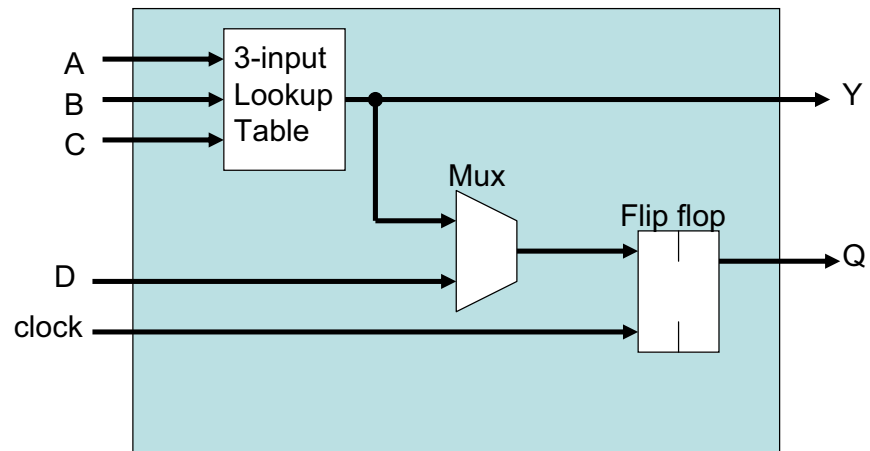


Figure A.7: Simple FPGA Configurable Logic Block composed of one 3-input Lookup Table [MC04].

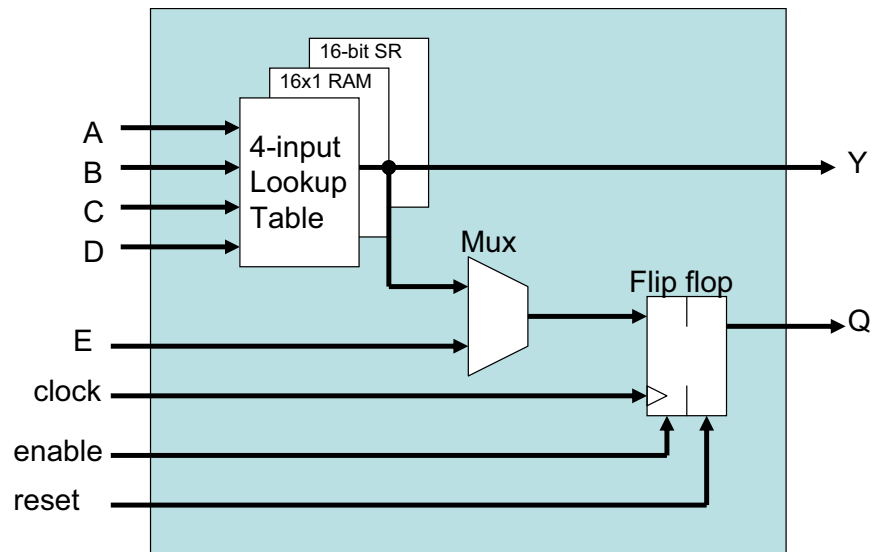


Figure A.8: Detailed FPGA CLB Design, showing the three uses of the 16x1 RAM [MC04].

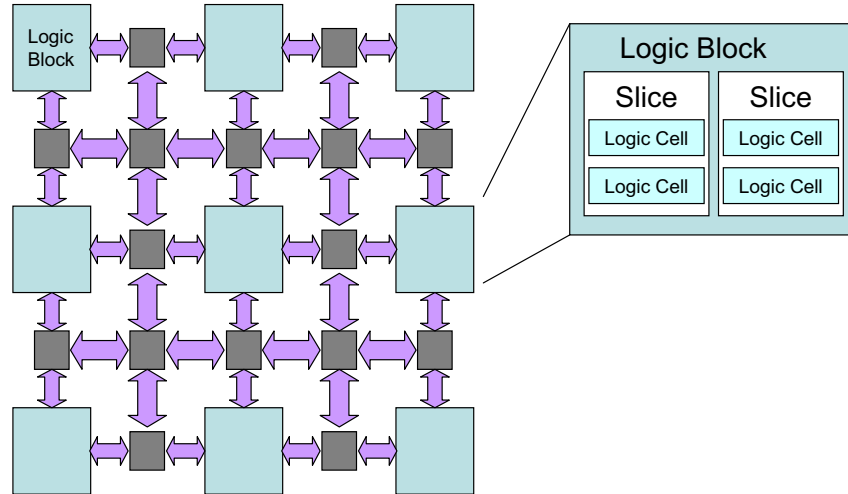


Figure A.9: Detailed FPGA CLB Design, showing component Slices and Logic Blocks [MC04].

A.9 combines several LUTs into a *logic cell*, which are combined into a *slice*, which are finally combined into the overall CLB.

Early FPGAs were formed from large arrays of CLBs with some additional logic at the peripheries to handle off chip communication and configuration functions. FPGAs incur significant area overhead due to the configuration memories and interconnect lines, and thus do not compete well with ASIC designs for large circuits. As process sizes declined and more space became available on the die, FPGA designers found additional functionality could be provided by embedding fixed-logic (i.e., ASIC) cores on the same chip as the programmable array, as shown in Figure A.10. The most common embedded cores are microprocessors and large memories. These cores take up much less area and typically operate much faster than the equivalent circuit implemented in CLBs in the FPGA fabric. Application designers use embedded cores to obtain functionality that would not otherwise be possible on the FPGA.

FPGAs provide a considerable amount of interconnect resources. Figure A.11 shows the types of routing available in a Xilinx FPGA [Xil05]. FPGAs have a hierarchical routing structure which provides maximum connectivity while minimizing delays. The FPGA in the figure has long lines, hex lines, double lines, direct connec-

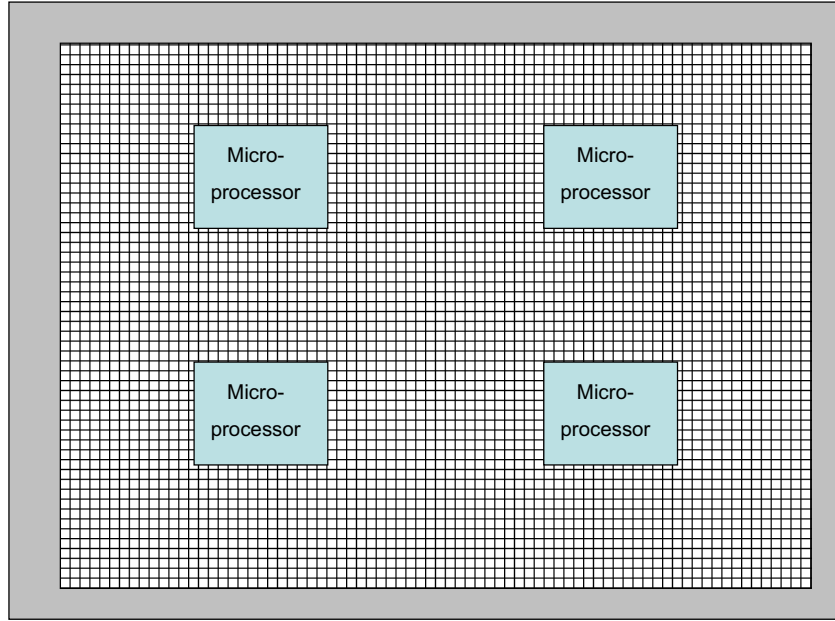


Figure A.10: FPGA Architectures containing embedded cores [MC04].

tions, and fast connects. In each channel in the CLB array, 24 long lines run the length of the FPGA. 120 hex lines connect every third CLB. 40 double lines connect every second CLB (as well as the adjacent CLB). 16 direct connections link adjacent CLBs in all directions (vertically, horizontally, and diagonally). Finally eight fast connect lines run internal to a CLB, connecting LUT outputs to LUT inputs. In addition to these signal lines, dedicated clock distribution lines exist as well.

A.1.2.2 FPGA Design Process. Designing an application circuit for a FPGA involves several complicated steps. Figure A.12 illustrates a design process beginning with schematic capture. Hardware Description Languages (HDL) such as the Very-High Speed Integrated Circuit (VHSIC) HDL, known as VHDL, are often used in place of schematic capture for large designs.

Once a gate level netlist is generated, the FPGA design tools perform the following functions:

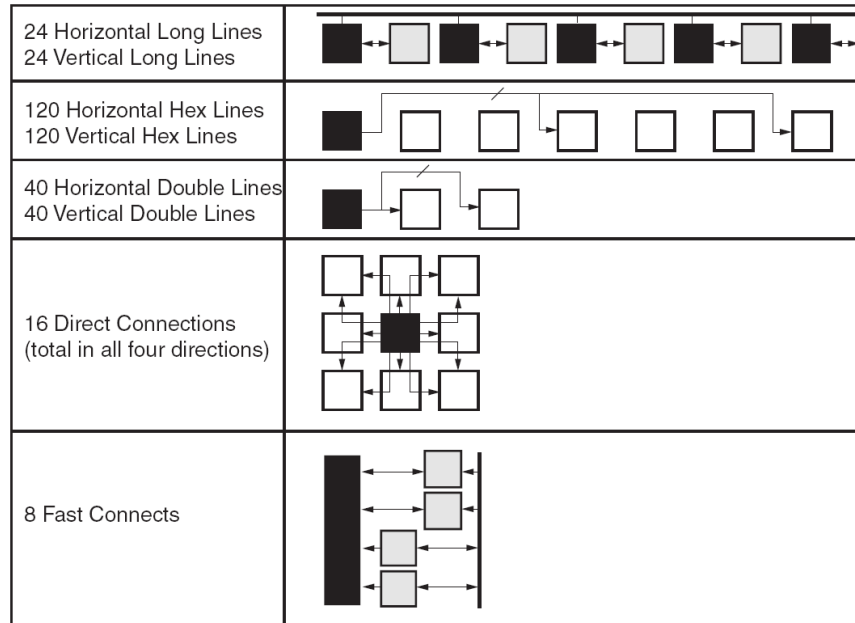


Figure A.11: Modern FPGAs have a large number of interconnect lines of varying lengths [Xil05]. Black blocks represent source Switch Matrices (SM). White blocks represent destination SMs. The grey blocks represent Configurable Logic Blocks (CLBs) in the top diagram, and slices in the bottom. Thus, every CLB in a row is connected to the horizontal long lines via switch matrices. The fast connect lines at the bottom connect slices inside the the CLB

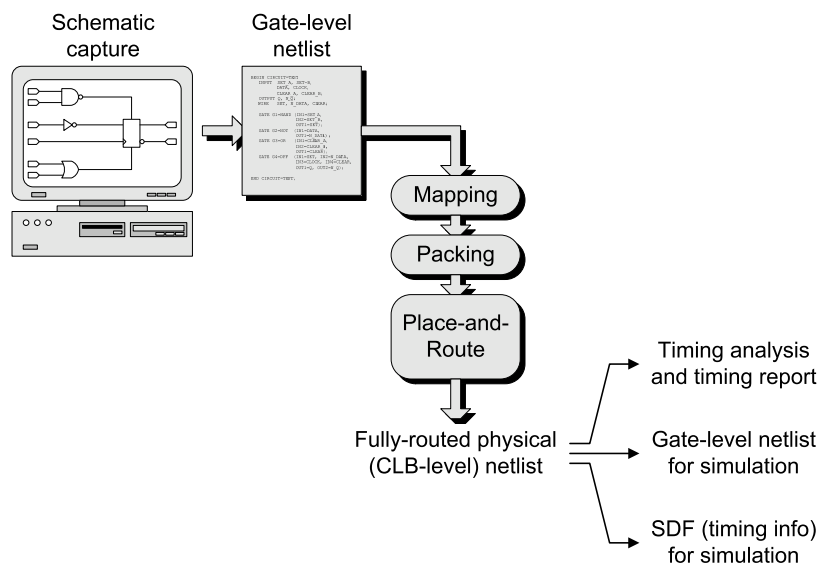


Figure A.12: Typical FPGA Development Process [MC04].

Mapping is the process of breaking apart the gate level design into LUT-level functions that can be mapped into LUTs.

Packing places the LUTs and registers together in CLBs.

Placement assigns the packed CLBs to locations in the physical FPGA array.

Routing assigns interconnect lines between the CLBs, connecting all of the CLBs in the design together, as well as to the input and output blocks.

Each of these steps can be time-consuming as the number of potential arrangements at each step is very large. This large number of permutations in the mapping of logic functions to CLBs, their placement in the FPGA, and the interconnect between the CLBs, results in a very large search space. Thus, the placement and routing process can require several hours to complete on a workstation level processor, and requires large amounts of memory. Research to improve placement and routing is summarized in Section A.1.4.

In a fault and defect tolerant FPGA, it is desirable to support *reconfiguration* of the application circuit to replace faulty devices. While limited reconfiguration may be done at the local level without rerouting the design, more global fault recovery will require dynamic routing. This may be done either on the FPGA or by an associated processor. Research in this area is discussed in Section A.1.4.3.

A.1.2.3 FPGA Configuration. The FPGA configuration file, also called the *bit file*, contains the bit values of all of the LUTs, the programmable interconnect registers, as well as special programming commands that control the load process [MC04].

In a simple FPGA, configuration registers are connected as a long shift register to support configuration, as shown in Figure A.13. The bit file is loaded in serial through a programming pin, and the bits are shifted through the entire device. The disadvantage with this method is that the entire FPGA must be configured at once.

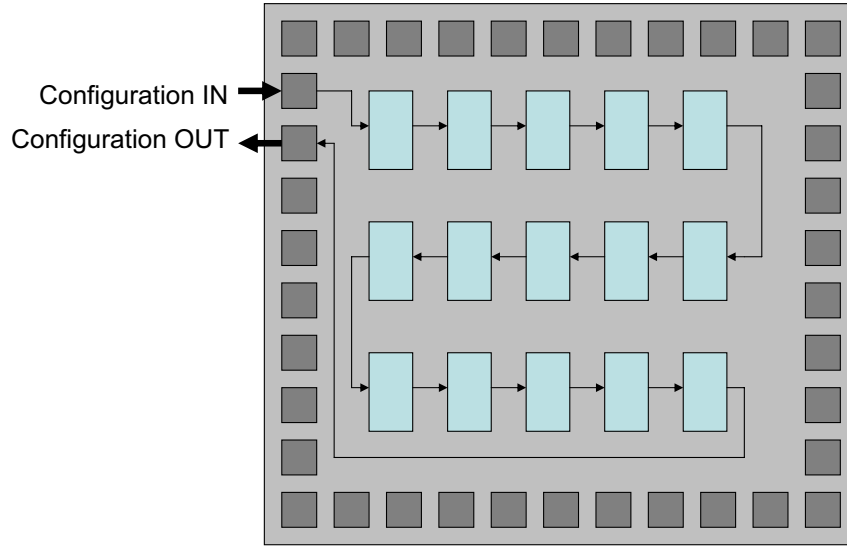


Figure A.13: FPGA Serial Configuration [MC04].

It is not possible to reconfigure portions of the FPGA. Modern FPGAs now support *partial reconfiguration*. Partial reconfiguration is discussed further in Section A.1.3.

The shift register view of the configuration memory is a simplification. To implement the memories as a shift register, each bit would be implemented using a flip-flop, chained together and clocked with a single clock. Since FPGAs contain large numbers of configuration bits (e.g., a typical FPGA in 2003 contained 25 million configuration cells [MC04]) and eight transistors required to implement a flip flop, the configuration memory alone requires 200 *million* transistors. To reduce transistor overhead, modern FPGAs use latches, which require only four transistors per bit, instead of flip flops. Since level sensitive latches cannot be used to create shift registers, FPGAs divide the bit file into *frames* of X bits. An X bit shift register is constructed from flip flops, and each frame is loaded in series. After a frame is loaded, the X bits in the shift register are sent *in parallel* to the X target configuration latches.

A.1.2.4 Limitations of Modern FPGAs. While providing tremendous capabilities to the designer, FPGAs are subject to several important limitations. The configuration and interconnect logic is a significant overhead, consuming a large area on the die and resulting in operating speeds that are often slower than ASIC-based

designs. The density of “user” devices on the FPGA is only 10-15% of an ASIC—a spatial redundancy factor of 7-10 [FNS01]. In addition, in most designs not all the CLBs can be used, leading to a ‘device packing’ redundancy multiplier (sometimes in the range of 1.5).

The SRAM memory used to store the configuration is volatile, and must be programmed before each use. The bit files used to configure the FPGAs are very large, and programming is time consuming. Early FPGAs required the entire FPGA to be programmed at once, requiring the application circuit to be stopped. Modern partially configurable FPGAs remove this restriction, allowing the rapid reconfiguration of parts of the FPGA while the rest of the device remains in operation.

In addition, current FPGAs are not fault tolerant and are very susceptible to single event upsets in the configuration memories, which can change the operation of the device or even cause it to suffer a permanent failure. Reconfiguration can provide fault tolerance, but current routing algorithms are very time and memory intensive. Although it has been proposed to allow the FPGA to dynamically re-route itself, research in this area is only beginning to produce results [LVT04a]. Dynamic routing is discussed in Section A.1.4.3.

Modern FPGAs are designed to provide the best performance for general purpose applications (i.e., those not requiring fault tolerance). As CMOS sizes shrink and soft errors become more common, commercial FPGAs will need to incorporate protections to provide reliable operation.

A.1.2.5 SEU Effects in FPGAs. FPGAs are more susceptible to single event upsets (SEUs) than ASICs. SEUs change the values of memory bits in flip flops, registers, and other memory devices. Since ASIC functions are hard-wired, the function of the device will not change due to an SEU. This is not the case with FPGAs, which are subject to the additional problem of errors in the configuration memories for the interconnections in the routing matrix as well as lookup table memories. A SEU in either location changes the function of the FPGA, and will persist until

the FPGA is reconfigured. This section describes how SEUs impact FPGAs as well as mitigation techniques. Many of these techniques are being used in space-based electronic systems. As device technology gets smaller and soft error rates increase, these techniques will be applied to mainstream applications.

Single Electron Events (SEE) impact the operation of a FPGA in a variety of ways. As with other devices, Single Event Transients can generate a short duration voltage pulse that is carried through the circuit. If the pulse arrives at the input to a flip flop during the clocking period, an incorrect value can be stored. Another common problem is the Single Event Upset (SEU), which is most commonly observed in the Configuration Memory Cells (CMC) [Xil03]. SEUs in CMCs change the operation of the FPGA. SEUs less commonly occur in the FPGA programming or control circuitry causing a Single Event Functional Interrupt (SEFI), which causes the FPGA to cease responding to configuration commands and become temporarily unusable.

SEUs are not normally capable of causing damage to FPGAs [Xil04a]. Although unlikely, a line connected to V_{dd} can be shorted to another line connected to V_{ss} . The short circuit between power and ground causes increased current flow, overheating the metal lines and causing failure. Another problem associated with short circuits is *driver contention*, in which multiple outputs are connected to the same bus. While not generally harmful to the FPGA, driver contention results in increased power consumption [CVR⁺03].

Although very unlikely, Single Event Functional Interrupts (SEFI) are a severe problem. FPGAs have circuitry to control the configuration of the FPGA (e.g., Power On Reset and the SelectMAP interface in Xilinx FPGAs) and its testing functions (i.e., the JTAG circuitry). An SEU in one of the Power On Reset flip flops can cause the FPGA to clear its entire configuration memory [Xil03, page 26], while a SEU in the SelectMAP interface used for parallel load can disable access to the configuration port and make it impossible to reconfigure the FPGA. The same problem can occur in the JTAG bit-serial configuration port and disable the entire FPGA. The problem

is not permanent, as correct function can be restored by applying control signals external to the FPGA or cycling the power. The probability of SEFI is very low, since the number of memory bits related to these functions is very small (e.g., only four for the Power On Reset function). Experiments show typical SEFI-inducing SEU rates for spaceborne FPGAs to be from 2×10^{-6} to 1×10^{-5} SEUs/device-day [Xil03].

Single Event Upset Rates (SEUR) in the configuration memory cells is much higher [CVR⁺03]. Typical SEU rates for spaceborne applications of the Xilinx Virtex-II 2V6000 FPGA are from 0.47 to 25 SEUs/device-day [Xil03]. In a worst-case scenario, a single SEU results in an observable error in the application circuit. The Mean Time Before Failure for the application circuit may be as low as 58 minutes. Fortunately, many SEUs in the configuration memory cells do not impact the application circuit. A typical application circuit depends on a fraction of the configuration memory cells. Even designs using all of the available CLBs will use a fraction of the routing resources and interconnect points in the switch matrices. In a typical design, only 10% of the CMCs in the switch matrices are used for connections between signal lines. SEUs in unused CMCs do not result in errors.

In addition to changing the function implemented by the CLB lookup tables, SEUs can change the interconnect. Four types of interconnect faults in switch matrices are [CVR⁺03]:

Open Connections in a signal line result in floating inputs to successive LUTs.

Antenna Connections reassign an output line to a different input line (e.g., instead of connecting A_{out} to A_{in} , it is connected instead to B_{in}).

Short Circuits connect multiple input lines to a single output.

No Effect. The SEU modifies an unused CMC in the switch matrix, resulting in no effect on the application circuit.

In addition to CLB lookup tables and interconnect switchboxes, SEUs affect other parts of the FPGA. Modern FPGAs often have large Block RAMs. The programmable Input/Output blocks can be altered, changing the function from input to

output and causing drive contention. Studies have shown the design of the typical I/O block is more resistant to SEUs than other parts of the FPGA and is less susceptible to SEUs [CVR⁺03].

SEU research falls into three categories: characterization and simulation of SEU effects on FPGA operation; diagnosis, avoidance, and recovery techniques that do not involve changes in the FPGA architecture; and techniques that modify the FPGA architecture.

SEU effects can be characterized by subjecting FPGAs to radiation [VCR⁺03, VSC⁺04, CVR⁺03, BRSV04]. Experiments confirm the predictions the CMC and LUT memories bits are most susceptible to SEUs, while I/O blocks are relatively immune. The goal is to develop an understanding of radiation effects on process technologies used to fabricate FPGAs and to combine this knowledge with simulation-based fault modelling techniques to develop SEU-immune designs.

Simulation-based techniques model the effect of SEUs on application circuit operation. Several projects investigate *fault-injection* techniques at the logic level [AMZE04, CVR⁺03, BRSV04]. These techniques estimate the probability a SEU will cause an error in the application circuit. The application circuit can be converted to a VHDL description using the Xilinx NCD2VHD tool [CVR⁺03]. The Native Circuit Description (NCD) file contains information about the configuration of the FPGA and its configuration memory settings. After conversion to VHDL, SEUs are inserted by flipping bit values of configuration memory cells. Simulating the circuit shows the effect of the SEU on the application circuit.

Combining radiation testing with simulation allows identification of the specific SEU causing a SEFI. By reading the configuration bitstream back out of the FPGA and comparing it to the original, the bits changed by SEEs can be identified, although it is not typically possible to do this until after several SEUs have occurred. Simulation of the detected SEUs can determine which caused the design to fail. Susceptible resources in the FPGA can be identified for radiation hardening.

Fault diagnosis and recovery techniques can be used in a FPGA-based system.

FPGA manufacturers recommend TMR and *scrubbing* [Car01, Xil03]. Modern FPGAs allow readback of their configuration bitstreams, which can be compared to the original bit file to identify SEUs in the configuration memory. These SEUs are corrected by reloading the original configuration, *scrubbing* the errors from the bitstream. Two scrubbing methods are read-compare-repair (i.e., called “closed loop”), and continuous periodic reconfiguration (i.e., called “open loop”).

Through the use of scrubbing, a high availability can be maintained. In an example spaceborne system, the Virtex FPGA is expected to be subject to a SEU rate of 1 per hour [CFBC99]. Configuration of the entire FPGA requires 40ms, while partial configuration of a single frame requires only $3\mu s$. The Virtex architecture allows the frame to be loaded in parallel without affecting the contents of flip flops and associated memories, allowing the user circuit to be refreshed with no impact on the operation of the user circuit. Open loop scrubbing can occur continuously at a rate of 25 times per second. Thus, on average as many as 90,000 correction cycles can be done between SEUs. With a 40ms recovery time to reconfigure the FPGA following an error, the availability of the FPGA is 99.9989%.

The previous example assumes the device is continuously refreshing its configuration. While in configuration write mode, the FPGA is slightly more subject to errors caused by an SEU in the configuration circuitry, although this probability can be decreased by reducing the refresh rate, depending on the required availability and expected SEU rate.

In addition to scrubbing, TMR can be used as a method of fault masking in user applications [Car01] and can be inserted into combinational and sequential circuits. Other techniques can reduce the impact of soft errors on clock distribution [Car01]. TMRTool is an automated tool to insert TMR into a non-redundant design [Car01].

Finally, SEU resistance of a FPGA can be increased by modifying the FPGA. Some FPGA manufacturers offer radiation-hardened FPGA product lines using Silicon-

On-Insulator fabrication and other special techniques. Researchers have proposed modifications to configuration SRAM cells [Wan04,SGV⁺04]. In addition, TMR in configuration memory was proposed by [BRSV04]. Fault tolerant FPGA designs are discussed in Section 2.6.2.

A.1.3 Dynamic and Partial Reconfiguration. Early FPGAs were designed to be configured at system startup, and not changed during application operation. The entire FPGA was configured at once, resulting in large bit files and long configuration times. Researchers studying reconfigurable computing and hardware/ software co-design needed faster configuration times as well as the ability to configure a portion of the FPGA while the rest remained in operation. The Xilinx XC6200 family, developed in 1995, was the first widely available FPGA to support dynamic partial reconfiguration. Since then, partial reconfiguration has become a common feature in FPGAs.

This section describes the partial configuration capabilities of the Xilinx Virtex-II series FPGA [Xil04b,Xil05]. Xilinx architectures are representative of commercial FPGAs. Different FPGA manufacturers have slightly different designs and capabilities. This section describes the capabilities and limitations of modern partially configurable FPGAs, and summarizes the design process.

A.1.3.1 Current Capabilities. The Xilinx FPGAs support column-based partial reconfiguration. After initially configuring a design, the FPGA can be partially reconfigured in groups of four slices (cf., Section A.1.2). All of the logic resources contained in the reconfigurable module area are available for use, including the I/O blocks, CLBs, tri-state buffers, Block RAMs, hard multipliers, and internal routing [Xil04b].

There can be any number of reconfigurable modules on the FPGA, limited only by the number of columns. Many designs implement a static section of logic, with one or more reconfigurable modules. Communication between modules must be through

Bus Macros, a “hard macro” placed at the interface between two modules to provide inter-module communication. Each bus macro can contain four signal bits. Wider signals are created by using multiple bus macros. The bus macros, placed manually during development, are not moved by the placement and routing tools. The bus macros are therefore a stable interface between the changing modules, allowing the developers to work with “black boxes,” minimizing interactions between the modules. All signals entering or leaving a module to other modules must be through the bus macros.

Two development processes are supported: *module-based* and *difference-based*. Module-based designs generate an entire bit file for each reconfigurable module. Difference based designs produce a much smaller bit file, describing only the differences between a module and its replacement. Thus, reconfiguration time can be very fast.

Reconfiguration can be done either through the serial boundary scan (JTAG) mode or Xilinx’ eight-bit parallel SelectMAP mode. Configuration is normally done in online mode, with the remainder of the FPGA remaining in operation during reconfiguration. Internal memory registers being reconfigured maintain their states and allow communication of data between a module and its replacement. This also allows for the configuration to be scrubbed to remove SEUs in an operating application circuit.

A.1.3.2 Limitations. While powerful, Xilinx’ approach to partial reconfiguration has several significant limitations:

- All intermodule communication must be through the fixed bus macros.
- The location of a reconfigurable module is fixed, and must always be in the same location (i.e., there is no capability to move the module to another part of the FPGA, although identical modules can be placed in other locations).

- Implementations must be designed so static portions of the design don't rely on the state of the module under reconfiguration. Explicit registers and handshaking may also be required.
- Signals transiting a reconfigurable module between two fixed modules must pass through the bus macros. As with the other signals, they are unavailable during reconfiguration of the module.
- Bitfile encryption cannot be used in partially configurable designs.
- Xilinx does not encourage changes in routing.

The last limitation is especially significant. Although possible, Xilinx' documentation for partial reconfiguration discourages routing changes [Xil04b]. Xilinx cites a risk of internal contention for routing resources. The architecture supports reconfiguration of I/O blocks (i.e., interface types such as low-voltage Transistor-Transistor Logic (TTL)), Block RAM contents, lookup table entries, and internal multiplexer settings. Therefore, a certain amount of functionality can be changed between configurations. Support for routing changes must improve to better support reconfigurable computing and dynamic routing applications.

A.1.3.3 Development Process. The development process for partially reconfigurable designs is much less automated than conventional fixed designs. Logic synthesis tools can generate the internal logic for each module, but the designer must manually place modules on the FPGA, as well as the bus macros between them. In addition, the Xilinx tools sometimes route signals across module boundaries, and the documentation advises the user to visually examine the routing in the design tools to verify the design. Signals which cross module (i.e., column) boundaries must then be manually corrected.

The typical development process is

1. Top level specification and floorplanning. Reconfigurable modules are assigned to specific areas on the FPGA fabric.

2. Bus macros are placed.
3. Black box modules are designed separately, using schematic capture or HDL synthesis.
4. Top level placement and routing is done for the entire design and its interfaces to the modules.
5. Each reconfigurable module is placed and routed.

The partial reconfiguration capabilities of Xilinx FPGAs are significant, but further improvement is necessary for advanced applications. The development process is not as automated as for a fixed design, requiring more designer planning and intervention. Entire columns must be configured at once, limiting the ability to make small changes to the design to correct faults. A fault in one CLB in a column can be corrected by reconfiguring the FPGA to move the module to another column. However, the entire column is removed from use, limiting the fault tolerance of the design. Improvements in FPGA design to better support fault tolerance are investigated as part of this research.

A.1.3.4 Configuration Improvements. A variety of techniques decrease the configuration time for partially configurable FPGAs. In a run-time reconfigurable computer, a large fraction of the total execution time is spent reconfiguring the FPGA to perform different functions. Techniques reducing configuration time will also be useful to a fault and defect tolerant computer. Most of these techniques have not been implemented in commercial FPGAs.

Configuration Compression compresses the bit file to minimize transfer time.

The bitstream is decompressed on the FPGA.

Context Switching replicates configuration memory cells (CMC) in the FPGA. An alternate configuration is loaded to the second group of CMCs while the first group controls operation of the FPGA application circuit. The configuration

Table A.1: Conventional Placement and Routing resource requirements [LVT04b]

Step	Memory Requirement	Time Requirement
Logic Synthesis	$\sim 10\text{MB}$	$\sim 1\text{ Min}$
Mapping and Packing	$\sim 10\text{MB}$	$\sim 1\text{ min}$
Placement	$\sim 50\text{MB}$	1-2 mins
Route	$\sim 60\text{MB}$	2-30 mins

contexts are switched very rapidly. In this way, configurations are loaded in the background while execution continues on the main context.

Configuration Prefetching is used at the system level to speculatively load a configuration before it is needed [Hau98]. This prevents the application pipeline from stalling while the new module is loaded.

Configuration Caching minimizes the transfer time to load bit files to the FPGA [LCH00, DST99]. Old bit file information is stored on fast memory near the reconfigurable array (typically on the FPGA itself).

Configuration Relocation/Defragmentation uses FPGA resources more efficiently by relocating and compacting application modules on the FPGA array [CLC⁺02]. Just as computer and hard drive memory get fragmented into small pieces as blocks are allocated and deallocated, fragmentation occurs in runtime reconfigurable FPGAs. Relocation moves operating modules to contiguous locations and combines unused resources into large areas useful for larger application modules. [Com99]

A.1.4 Placement and Routing. This section examines the two most difficult tasks of the FPGA design process: placement and routing. As shown in Table A.1, the placement and routing steps require the most memory and CPU cycles to perform, and are the most difficult to implement on-chip in a fault and defect tolerant computer. Conventional algorithms for routing are described, as well as research to implement a router on the FPGA itself.

A.1.4.1 Placement. Placement algorithms attempt to minimize a cost (objective) function, while complying with placement constraints. Typical constraints include [Cha94]:

- Locking certain I/O blocks as required by the user
- Aligning tri-state buffers that belong to the same bus along a long line

Typical cost functions include

- Minimization of total interconnect wire lengths
- Minimization of the number of configuration bits required
- Congestion reduction to ensure routability
- Minimization of connections across different regions on the chip

Several of these costs are difficult to estimate accurately prior to routing, and thus estimates are used. For many FPGA design tools, the entire process is done in two phases: a fast initial routing based on estimates, then further refinement based on updated estimates.

One placement strategy uses a *mincut-based placement* algorithm followed by *simulated-annealing based placement* [Cha94]. In mincut-based placement, locality is exploited by clustering together portions of the circuit that are closely related. The design is partitioned into two or more partitions, attempting to minimize the number of lines connecting the two partitions. Each partition is itself partitioned, and the process repeated. Finally, a minimal size partition is reached (related to the size of the CLBs). The partitioning shows the required communication between CLBs.

Simulated annealing can overcome local minimums in the cost function that may result from using a greedy algorithm. Simulated annealing starts with a feasible solution and a *temperature* function, initially set very high. The solution is modified randomly (e.g., through pairwise exchange of placement locations) and its cost function recomputed. If the cost function is better than the original, the new solution

is adopted. Even if the cost function is worse than the original, the new solution is adopted with some probability, which depends on the current temperature and the difference between the previous cost function and the new cost function (ΔE). The probability of accepting the new placement, $P(A)$, is

$$P(A) = e^{-\frac{\Delta E}{T}}. \quad (\text{A.1})$$

New solutions are generated at a particular temperature until no further improvements are observed. Once this equilibrium is reached, the temperature is dropped and new solutions are tried again. At high temperatures, $P(A)$ is close to one, so most exchanges are accepted. As the temperature decreases, only small changes in the cost value are accepted. The process is completed when no further improvements are observed despite further reductions in temperature.

A.1.4.2 Conventional Routing. Conventional routing algorithms are time and memory intensive. The Coarse Graph Expansion (CGE) algorithm [BRV92] requires between 1.5 - 7 MB of memory and 215 seconds to route designs with only 100-586 logic blocks (and between 400-2100 connections). Table A.1 showed typical placement and routing resource requirements for larger FPGAs [LVT04b].

FPGA routing is typically done in two phases: *global routing* and *detailed routing* [Cha94, BRV92]. Global routers assign loose paths to the nets, determining how they navigate around the CLBs. The loose paths determine which areas a net will traverse to get to its destination(s). The routing areas can be channels, segments of channels, or switch boxes. The objective of the routing assignment might be to shorten the path lengths, avoid congestion, or use fewer programming elements to decrease delay in reconfiguration. Many routing algorithms exist, but in most FPGA routers the key goal is to minimize the delay along the critical path. After global routing has been completed, the detailed router assigns specific tracks to nets within the routing regions and makes the final connections to the CLBs.

Many common routing algorithms are derivatives of *maze routing* [LVT04b, Cha94], first proposed in [Lee61]. Maze routers place each net using Dijkstra’s shortest path algorithm. The maze router attempts to find the shortest path between two points subject to the restriction that the route must follow vertical and horizontal paths, and the routing areas includes obstacles that may force the router to detour. Although computationally expensive, Lee’s algorithm finds the shortest path between nodes (if one exists). However, since the router operates on one net at a time, the order in which nets are routed is important. Due to congestion caused by previously assigned nets, it is possible that the router may be unable to find a path for a net to one or more of its destinations. One variation of the maze algorithm intended to overcome this problem is Maze Routing with Rip-Up and Retry [Cha94]. In this case, the router enters a *rip-up mode* to complete the connections. In rip-up mode, previously routed nets blocking the current net are removed to eliminate obstacles. Ripped-up nets can then be rerouted.

Another routing algorithm is *Pathfinder* [EMHB95]. This algorithm introduced *negotiated congestion*. During each routing iteration, all nets are routed using shortest paths, without regard to other nets (i.e., overuse of resources is allowed). After the initial routing step is performed and two nets are found to use the same routing resource, the router updates the cost of congested resources based on the amount of overuse. All routes are then ripped up and routing is done again. The *Versatile Place and Route* (VPR) tool uses a modified pathfinder algorithm [BRM99, BRM03].

Placement and routing are very complex processes and the algorithms used greatly impact the performance and resource requirements (i.e., size) of the completed design. It is also possible a routing algorithm will be unable to route the design. The probability that a router can successfully route a design is called *routability*. Three dominant factors affect the routability of an FPGA design [Cha94]:

- Pins per logic cell ratio, γ
- Pins per net ratio, β

- Average wire length, L

The pins per logic cell ratio measures the amount of traffic entering and leaving a CLB. Pins per net measures the degree of branching (i.e., fanout) of a multipin connection. Both of these ratios are dependent on the application design, as well as on the architecture of the FPGA. Average wire length depends mostly on the routing tools.

These factors can be used to estimate the routability of a design in a homogeneous two-dimensional array [Cha94]. If all the nets are point-to-point connections (i.e., $\beta = 2$), the average *channel width*, W , or the number of parallel interconnect lines, is

$$W = \frac{\gamma L}{2}. \quad (\text{A.2})$$

For multi-pin nets, the equation is modified to become

$$W = \frac{1}{2} \left[\gamma \cdot \left(1 + \frac{\beta - 2}{\beta} \right) L \right]. \quad (\text{A.3})$$

Using these equations, the router can quickly estimate the relative interconnect requirements and determine if routing is likely to be successful. For example, after placement, $\gamma = 6.15$ and $\beta = 4.64$. The router initially estimates an average wire length to be $L = 1.5$ segments. The estimated channel width requirement from A.3 is $W = 7.245$. If the FPGA architecture only has 5 lines in each channel, routing is unlikely to be successful. Routability estimation will be important in dynamic routing (cf., Section A.1.4.3, in which the processing resources to perform routing are limited).

Many routing algorithms and techniques propose improvements in routing time/memory requirements, as well as performance of the final circuit. One technique partitions the routing problem between multiple workstations to run in parallel [CS97].

Others attempt to do routing on the FPGA itself, either prior to operation or during runtime. Dynamic routing is discussed in the next section.

FPGAs can be designed to support faster routing. As described in [LVT04a], the Programmable Logic and Switch Matrix (Plasma) architecture was designed to allow the entire FPGA to be routed in three seconds. This was accomplished through the use of very large amounts of interconnect, arranged in a hierarchical structure. While routing is done very quickly, the Plasma architecture requires an even larger amount of redundant interconnect than the standard FPGAs.

FPGA routers can be used for defect tolerance. An algorithm was developed to route around faults in the programmable interconnect in antifuse-based FPGAs [RN95]. Through several test cases, 100% routing was achieved given sufficient redundant interconnect. Three key factors determine routability under faulty conditions: the existing placement (defining routing requirements), the channel architecture (defining available routing resources), and the routing algorithm.

A.1.4.3 Dynamic Routing. Dynamic routing refers to routing performed while the FPGA is in operation, either by the FPGA itself or by an attached processor. Dynamic routing can simplify the distribution of combined hardware/software applications running on reconfigurable computers, as well as to provide fault and defect tolerance.

A dynamic router should have several key features:

- Minimal memory requirements.
- Fast runtime.
- Low-level control over individual wires.
- Hierarchical representation of the FPGA so localized routing can occur in parallel.
- The ability to incrementally add and remove connections.

Incremental Routing works without any prior knowledge of the mapped circuit's netlist [EB98]. To keep the memory size small, the algorithm keeps the “window” of the FPGA available for re-routing relatively small (i.e., limiting the options and sacrificing routability to keep memory requirements small). Incorporation of unused logic blocks in the design increases routability by lowering the density of logic blocks to routing resources.

The algorithm is purported to be sequential and compact, making it well suited for execution on the FPGA itself. However, it does not guarantee routability, and it is possible that the greedy algorithms will not result in a solution. Thus, the algorithm could not be the sole method used in a fault and defect tolerant computer.

Conventional routers use a flat representation of the routing segments and switches, which consumes a large amount of memory [KM02]. For dynamic or on-chip routers, a better representation is desirable.

JBits is a runtime reconfiguration tool from Xilinx containing a runtime router called JRoute [KM02]. JBits uses a wire database to store wire connectivities with Java objects. The objects are stored in a device generic manner to reduce storage requirements. Repetitive structures are included by reference, whereas conventional routers commonly repeat descriptive information for each occurrence. Inter-tile connectivity information is not stored statically in JBits, but is generated dynamically during router operation which reduces memory requirements at the expense of additional time to generate the connectivity data. Automatic dynamic routing is supported by JRoute/JBits. Defect tolerant routing is supported, as JRoute has the ability to specify which wires and CLBs to use in the routing process. Thus, defective resources can be removed from the list of usable resources.

The Riverside On-Chip Router (ROCR) paradigm provides a single hardware specification which is dynamically routed by each FPGA before use. Thus, a single software and a single hardware specification can be distributed. Just-In-Time (JIT) compilation is used [LVT04a, LVT04b]. The project uses an “on-chip” router

for FPGAs. Modern electronic appliances (e.g., cable boxes, satellite decoders, cell phones, etc.) often contain FPGAs. Applications for these devices often combine software running on a conventional processor with custom application-specific hardware running on FPGAs (cf., Section A.2). The application is distributed as two parts: software code and an FPGA bit file. Companies often field different hardware variants of an appliance, each containing different FPGAs. In this paradigm, the application’s hardware configuration must be regenerated for each FPGA in use, complicating the distribution process.

ROCR is based on the *Versatile Place and Router* routability-driven router. The algorithm, illustrated in Figure A.14, constructs a cost model consisting of a basic cost, updated with historical congestion and current congestion costs [LVT04b]. Routing is done between switch matrices in the interconnect fabric. The algorithm routes nets between switch matrices using a greedy, depth-first routing algorithm. This technique claims to be faster than the traditional breadth-first method used in VPR, but requires the addition of an “adjustment cost” to force ROCR to re-route illegally mapped nets using a different initial path [LVT04b]. As in the Pathfinder algorithm, nets are initially routed without regard to contention. If congestion is discovered, the ROCR router rips up *only the illegal routes*. This improves routing time considerably.

ROCR differs from conventional routers by using a smaller resource graph. The regular design of the CLB-switch matrix connections allow routing between CLBs to be represented instead by routing between switch matrices (which are fewer in number than the surrounding CLBs). The result is a smaller directed resource graph that can be routed faster and with less memory [LVT04a]. The nodes of the graph represent switch matrices, and the edges represent interconnect resources. Two types of edges are used to denote short and long routing lines. Similar to other FPGA architectures, the FPGA design used with ROCR has short lines connecting adjoining switch matrices used for local routing, while long lines skip every other switch matrix, and are used for longer distance paths.

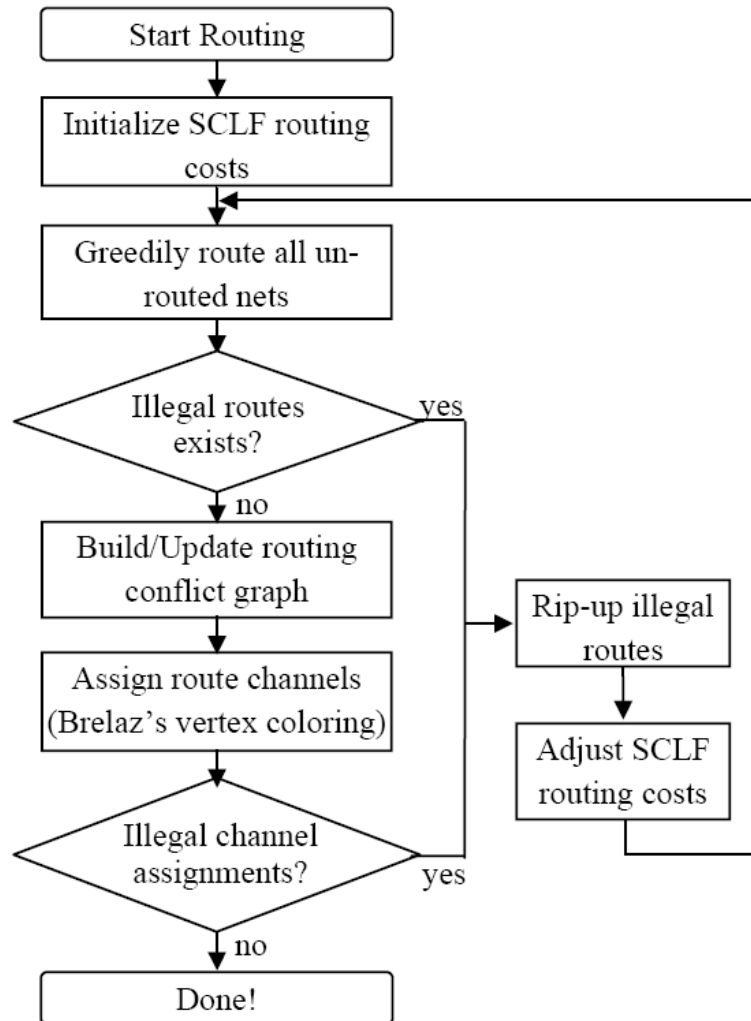


Figure A.14: The Riverside On-Chip Routing algorithm [LVT04a].

Table A.2: Comparison of the resource requirements between VPR and ROCR [LVT04b]

	VPR	ROCR
Memory Requirement	4-57.3MB	~ 3.6 MB
Runtime Requirement	0.5 – 1428 sec	0.2 – 13.8 sec

The ROCR routing algorithm significantly reduces runtime and memory requirements, as shown in Table A.2. ROCR was compared to VPR for 17 benchmark circuits and on average, ROCR was 10 times faster than VPR and used one tenth of the memory [LVT04b]. ROCR produced designs only slightly less efficient than VPR, utilizing 10% more routing resources. Performance of the ROCR circuits was better than VPR, with critical paths 10% shorter than those routed by VPR.

A.2 Reconfigurable Computing

Field programmable gate arrays fill a gap between ASICs and PLDs by providing the capability to change hardware function to match a particular application. This capability can be extended to the computer architecture itself by providing the computer with the capability to change its hardware to suit applications. This concept is called *Reconfigurable Computing*.

Design engineers have several options when implementing an application with digital systems. At one end of the spectrum, the application can be implemented entirely in software, to be executed on a general purpose microprocessor (GPP) . Microprocessors provide a variety of relatively simple instructions and capabilities. From these simple instructions, complex programs can be constructed to implement a wide variety of applications. Flexibility comes at a cost, as general purpose processors provide limited hardware support for any particular application. At the other end of the spectrum, the application can be implemented entirely in hardware as an application specific integrated circuit (ASIC). In this case, the hardware design is tailored to the specific application to provide maximum performance. However, the ASIC has limited use for other applications. In the middle of the spectrum, Application

Specific Instruction Processors (ASIP) add specialized instructions and capabilities to a general purpose processor to increase performance on a specific type of application. Digital Signal Processors (DSP) are a type of ASIP, providing extra hardware support for common signal processing operations such as the fast fourier transform. Another example is graphics processors found in modern video cards.

While ASIPs provide some performance gain over general purpose processors, these benefits are only gained for a set of pre-planned operations. With the development of the FPGA, designers have a hardware device that can be reconfigured in any number of ways to support a wide range of applications. These reconfigurable computers combine the high performance of ASICs with the flexibility of a general purpose processor.

By definition, a reconfigurable computer contains some amount of programmable logic. The functionality of this programmable logic can be changed during the operation of the computer to suit the particular application being executed. The entire architecture, or only a portion, may be configurable.

Reconfigurable computing is applicable to fault and defect tolerant computing. Reconfiguration is one method of fault tolerance. If a portion of the hardware used by an application circuit fails, reconfiguration can move the circuit to a different part of the chip. Thus, a fault tolerant computer that uses reconfiguration is itself a reconfigurable computer.

This section provides a brief introduction to reconfigurable computing. The motivations for reconfigurable computing are introduced, as well as typical applications. Common system architectures for reconfigurable computers are described, and an application development process for these systems. Development of mixed hardware and software applications has created an entire research field, that of *Hardware/Software Codesign*. Finally, several challenges involved in reconfigurable computing are described, with emphasis on how they impact fault tolerant computing.

A.2.1 Applications. Reconfigurable computing has drawn broad interest for a variety of applications. The most widely known applications are signal processing and cryptography. These applications often include operations that are very parallel in nature and well suited to implementation in hardware. The reconfigurable computer implements a large number of simple processing modules in parallel, and can operate on much larger amounts of data than a serial program running on a general purpose processor. While an ASIC implementation of the same circuit is often smaller and faster than the FPGA implementation, the FPGA can be reconfigured later to perform other applications or to fine tune the algorithm to match a changing problem.

Many space applications take advantage of the ability to reconfigure the hardware during operation. Reconfigurable computers can be used on long term space missions to provide better performance than a purely software implementation of an application. Design changes can be sent to the spacecraft in flight to implement different applications. In addition, should part of the system fail in flight (e.g., due to radiation or other damage), the FPGAs can be configured to replace damaged ASICs (albeit at lower performance).

A.2.2 Typical Architectures. Reconfigurable computers support a variety of architectures. Early reconfigurable computers attached an FPGA to a conventional computer via a communications link. Later designs brought the configurable logic closer, first as a coprocessor, then as a part of the main processor itself. The primary difference in architectures is the amount of reconfigurable logic in the design, and the extent to which it is integrated with the main processor.

The first reconfigurable computers were loosely coupled to a general purpose computer. A typical architecture is shown in Figure A.15. This architecture is sometimes called the static logic model. FPGA configuration bit files can be very large, and configuration time can be significant. In static logic architectures, the FPGA is configured prior to application operation without reconfiguration during operation. Communications bandwidth is limited, and thus loosely coupled architectures are

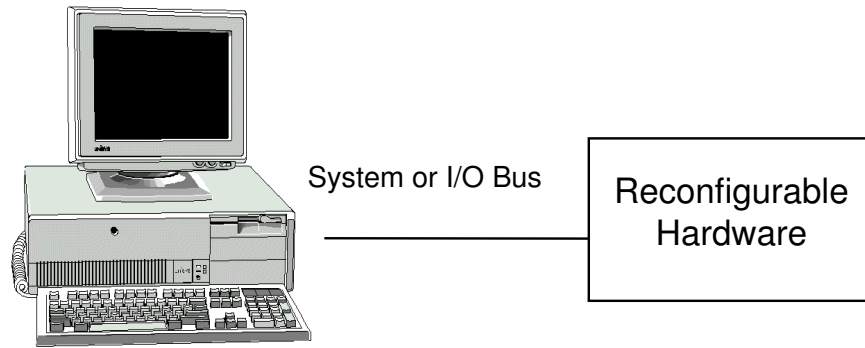


Figure A.15: A loosely coupled reconfigurable computer has limited communication with the host computer's memory. [Roe97].

best suited for applications requiring limited communication with main memory or the primary processor.

An example of a static logic system is the PerLe system, developed by DEC Paris [BRV89]. The system was based upon a 5x5 array of Xilinx 3090 FPGAs, providing roughly 150,000 logical gates to the user. Applications for the system include RSA cryptography, Laplace transforms, long multiplication, and a stereoscopic vision system. In 1990, PerLe set a speed record for RSA cryptography using 512-bit keys, delivering roughly ten times the performance of a custom VLSI implementation. This performance was largely a result of the ability to easily customize the algorithm on the reconfigurable hardware.

Later reconfigurable computers brought the FPGA onto the motherboard as a coprocessor. As shown in Figure A.16, the FPGA is connected to the memory bus, significantly improving the communications bandwidth between the GPP and the reconfigurable logic. Typical applications for these architectures combine software code executing on the general purpose processor with an FPGA bit file. Portions of the application are partitioned, usually by hand, between the GPP and the FPGA. Those portions of the application which would benefit from hardware implementation were placed on the FPGA, while the remainder is left as software. Interface code and logic is inserted to allow communication of data and results. An example of the

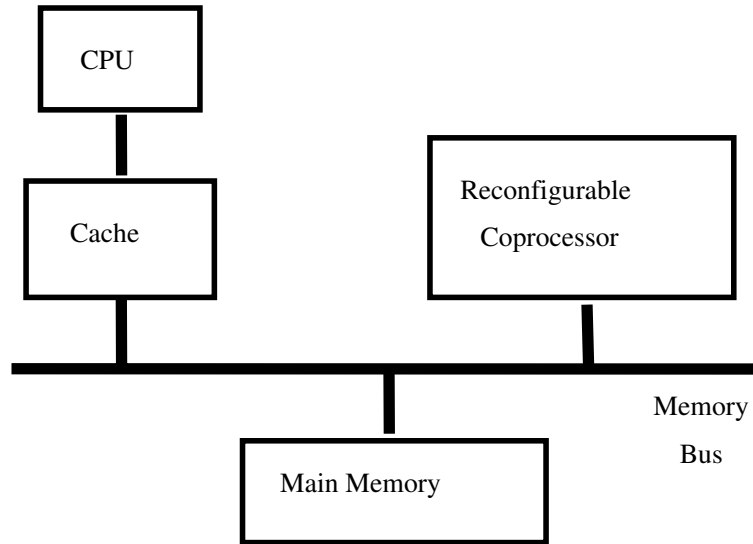


Figure A.16: A reconfigurable coprocessor is connected to the computer's primary memory but is not a part of the CPU itself [Roe97].

coprocessor architecture is the CHAMP system, developed at AFRL. Other examples include the PRISM system [AS93], the Transmogripher system [Gal95], and Virtual Computer Corporation's EVC-1 [CTS95].

A fault tolerant computer architecture could use a coprocessor-based design for use in spacecraft. One or more redundant FPGAs would be connected to the memory bus. If one of the FPGAs used in the system fails, its function can be moved to one of the redundant FPGAs.

Another notional reconfigurable architecture is shown in Figure A.17. In this design, the reconfigurable coprocessor is incorporated onto the same microchip as the microprocessor.

The programmable logic can be brought onto the main processor itself. A reconfigurable processor pipeline is shown in Figure A.18. In this design, the FPGA unit implements custom operations such as the bit reversal operation, commonly used in cryptography and signal processing. Termed a *Dynamic Instruction Set Computer* (DISC), each application loads its own custom instructions [WH95]. As the configura-

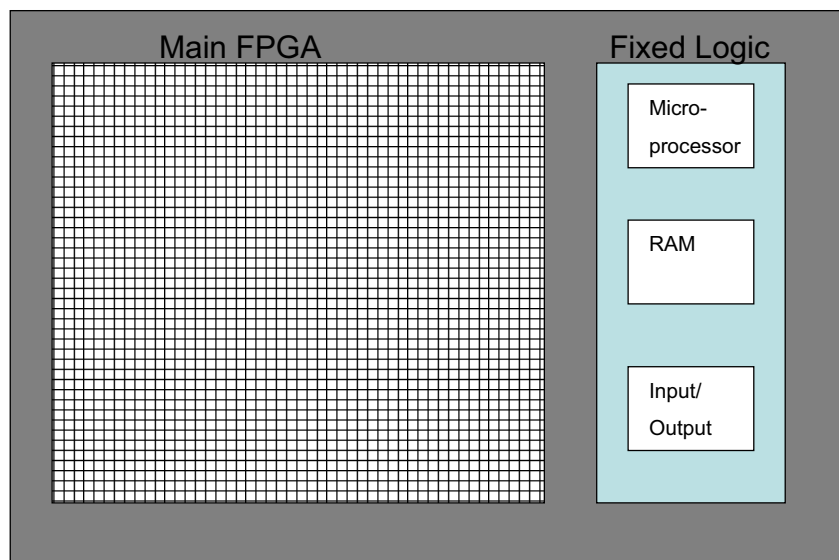


Figure A.17: RC hybrid processor containing both a reconfigurable array and fixed logic [MC04].

tion bit files can be large, configuration of the FPGA unit requires many clock cycles, limiting the number of instructions that could be used. Advanced FPGA designs providing configuration caching can store multiple configurations at once, enabling rapid switching between configurations.

As a final step in integration, the entire processor could be implemented on the configurable logic fabric. Due to the limited number of logic gates that could be implemented on early FPGAs, it was not possible to implement an entire processor on a single FPGA. Modern FPGAs provide the equivalent of more than a million gates, and can implement entire processor designs. In addition, the nanoscale technologies discussed in Section 2.3.1 will provide even larger numbers of devices. In the future, the reconfigurable processor paradigm is likely to be more widely used.

A.2.3 Application Development. Application development is the most significant challenge in reconfigurable computing. Most architectures combine custom hardware implemented on a FPGA with software code executing on a general purpose

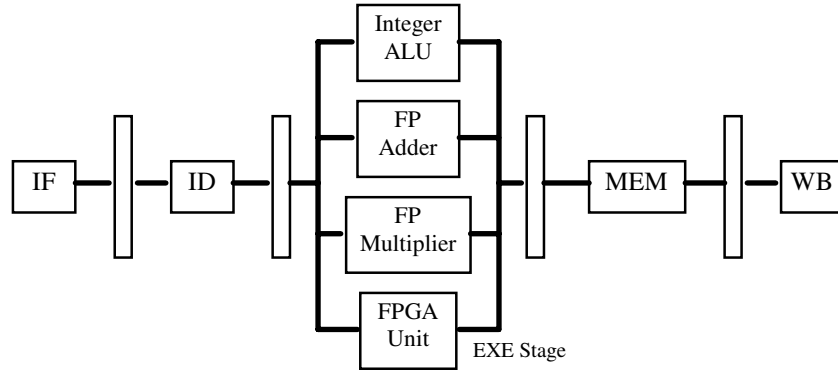


Figure A.18: In a reconfigurable processor, the program-mable logic is incorporated into the processor itself, possible as a configurable execute unit (shown). In other implementations, the entire processor may be implemented on the configurable logic [Roe97].

processor. Simultaneous design of a software application with supporting hardware is called *hardware/software codesign* .

Initially, most codesign development was done by hand. Computationally intensive portions of the application are identified. Hardware implementations of these sections are developed using schematic capture, or logic synthesis using a hardware description language. Interface code is added to the software application to communicate with the hardware processor, and interface logic is added to the hardware modules to read input data and write back results. This process is labor intensive and requires the designer to have a detailed knowledge of the underlying hardware architecture.

Research efforts have attempted to simplify the development of hardware/software applications. The main goal is to not require the application designer to have detailed knowledge of the underlying hardware architecture and to create portable applications. The problem has proven difficult for two reasons: it is often difficult to identify the sections of code best implemented in hardware, and it is difficult to describe hardware functions with conventional high level languages.

High level languages provide constructs and operations based upon general purpose processors, and are not well suited to describe hardware operations. Many operations which have direct hardware implementations (e.g., reversing the order of bits of an operand) are described by a long series of simple instructions. Automatic replacement of a sequence of instructions with a simple hardware structure is difficult [Roe97].

At the same time, hardware description languages (HDL) are not ideal solutions for application development. Behavioral HDLs describe hardware operations, not object-oriented data structures and other concepts used in modern programming languages. For this reason, many researchers have proposed hybrid hardware description languages that more closely resemble high level programming languages. These languages would be similar to C or other HLL, and would be easily learned and used by the software developer. At the same time, the hybrid HDL would concisely describe hardware structures not possible with a conventional HLL. Two examples of C-based hybrid HDLs are the Transmogripher C project at the University of Toronto [Gal95], and the Spyder project at the Swiss Federal Institute of Technology [IS95]. A more recent approach based on standard Java is the JHDL project (short for “Just Another Hardware Description Language”) [BH98, HBH⁺99]. With hybrid HDLs, the partitioning of the application between hardware and software must still be done by hand.

Reconfigurable compilers would allow the specification of an application entirely in a high level language [Roe97]. The compilation process is illustrated in Figure A.19. Runtime profiling automatically identifies computationally intensive portions of the application. These sections are converted to a hardware description language representation and then synthesized for use in the FPGA. The output of the compiler is a software executable file and hardware bit files to configure the FPGA.

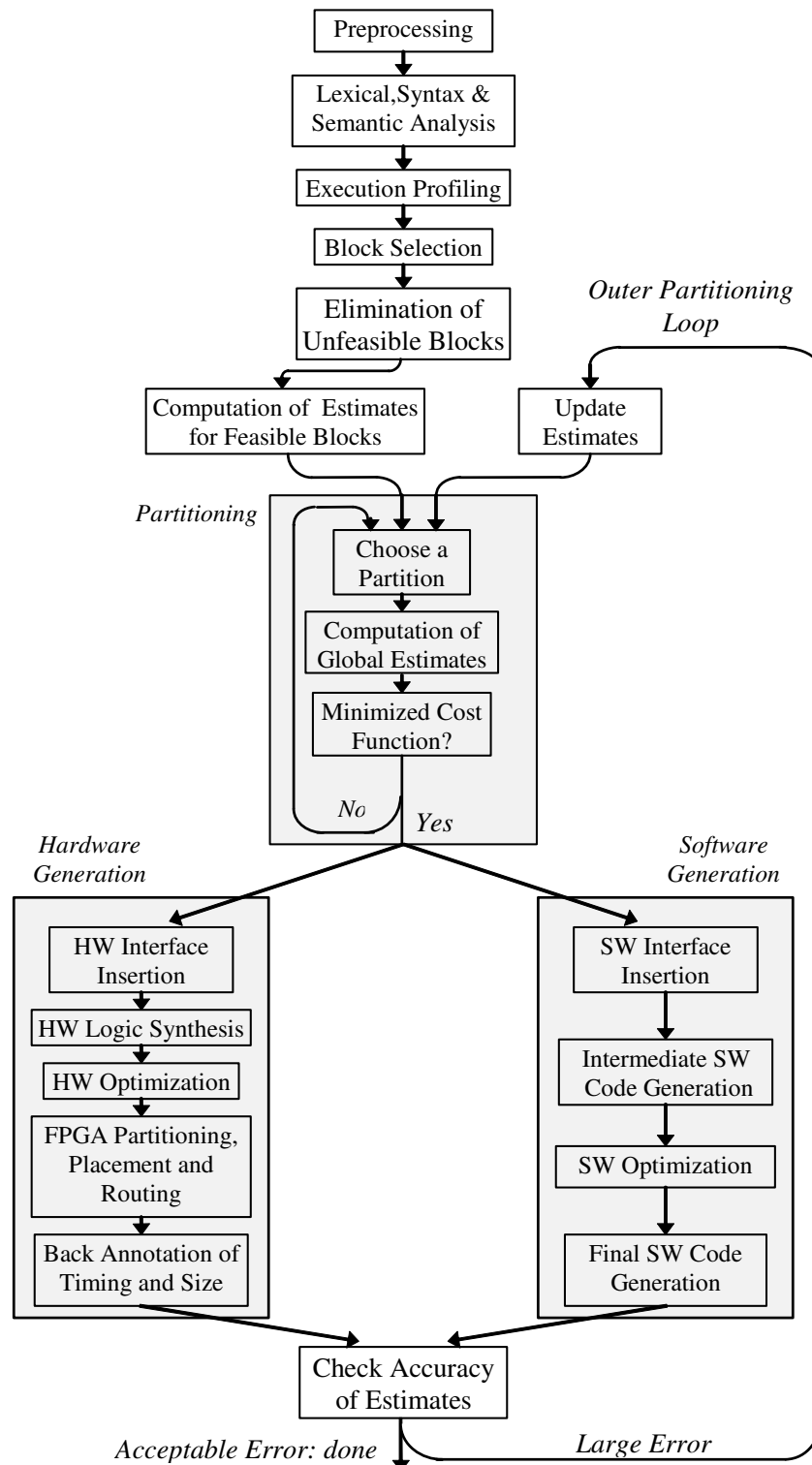


Figure A.19: Stages in a Reconfigurable Compiler [Roe97].

A.2.4 Limitations. The three major limitations of reconfigurable computers are device density, lack of development tools, and configuration overhead. All three of these limitations are also relevant to defect and fault tolerant computing.

As discussed in previous sections, FPGAs require a considerable amount of chip area to implement the configuration memories and interconnect lines. Thus, the number of logic gates that can be provided to the application developer are much less than that available with an ASIC. This limitation is becoming less of an impediment as device density increases. In the future, molecular crossbar FPGAs and other architectures may provide massive amounts of configurable logic to the designer. The configurable logic may be used to implement custom hardware functions, fault tolerance hardware, or a combination of both.

Application development is a significant problem. To be widely adopted by applications programmers, detailed hardware knowledge should not be required. The details of the hardware should be handled by the compiler and the operating system. In a fault tolerant computer, reconfiguration should be handled by the chip itself, or by the operating system, with minimal awareness by the application developer or the user of the software.

Finally, configuration overhead has been a limiting factor for reconfigurable computing. Early FPGAs did not support partial reconfiguration, requiring the entire FPGA to be taken offline and reconfigured at once. Many modern FPGAs now support online partial reconfiguration, but reconfiguration time can still be a significant portion of the overall execution time for an application. To compete with software only implementations, the overhead involved with reconfiguration must be minimized. Figure A.20 illustrates the components of application runtime for a reconfigurable computer. While the hardware runtime is less than a software only implementation, the reconfiguration overhead may actually result in a slower implementation.

Fault tolerant reconfigurable computers must minimize fault handling overhead to compete with modern CMOS implementations. Dynamic routing, discussed in

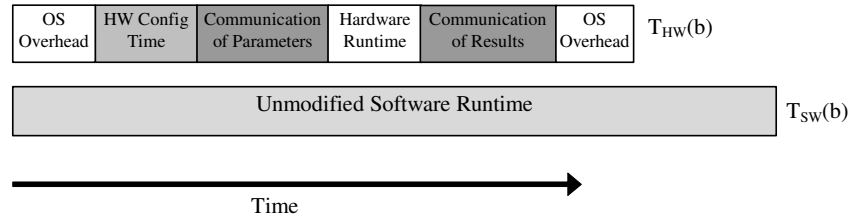


Figure A.20: Runtime Computation for a Reconfigurable Computer. The runtime of a RC application must include configuration time, and thus can be longer than the runtime of a software-only application. A fault tolerant computer would include similar overhead, which must be kept small to compete with traditional CMOS implementations [Roe97].

Section A.1.4.3, is computationally intensive. However, if the failure rate for the device is sufficiently low, the performance penalty for dynamic routing may be acceptable. If failure rates are high, the dynamic router may be called often, and the overall performance penalty will likely be unacceptable.

Appendix B. Long Term Group Research Goals

B.1 Four Research Phases

The primary focus of this research is to determine how reliable computing can be accomplished using emerging device technologies that are more defect and fault prone than modern silicon CMOS. Is it possible to build a reliable large-scale circuit such as a microprocessor using devices with defect probabilities orders of magnitude worse than CMOS? What minimum performance characteristics must these devices possess to be a viable alternative to CMOS? This research addresses an emerging need in aerospace electronics to provide reliable information technology in the hostile environments future warfighters will face.

Building a fault and defect tolerant computer requires several enabling technologies, discussed in Chapter II. The most important will be:

- Hardware fault tolerance techniques
 - Fault masking
 - Fault detection and diagnosis
 - Fault recovery
 - Reconfiguration
- Programmable logic device technology
 - Fault tolerant PLD architectures
 - Fault detection and diagnosis in PLDs
 - Partial and dynamic reconfiguration
 - Online and On-Chip Routing (OCR)
- Computer architecture support
 - System level fault recovery
 - Dynamic routing support

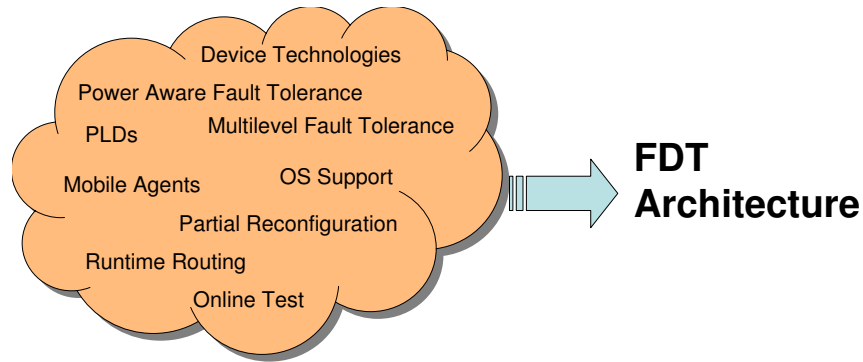


Figure B.1: Many enabling technologies will be combined to create a system architecture for a fault and defect tolerant computing.

- Operating System (OS) support
- Device technology

Reliable computing is a large problem, and addressing it begins by breaking the problem up into several research phases. The first phase defines a FDT system architecture and determines its required capabilities. The large collection of fault tolerance concepts shown in Figure B.1 are synthesized into a coherent, multi-level strategy for reliable system operation. From these required capabilities, limitations in supporting technologies can be identified. Finally, supporting technologies can be developed or improved.

Three additional research phases are proposed for follow-on research by a new AFIT fault and defect tolerant computing group. Figure B.2 shows all four research phases. Since reconfiguration will be a key capability for a FDT computer, the FDT computer will likely be implemented on a reconfigurable mesh similar to a FPGA. Given the limited fault tolerance capability of modern FPGAs, the second phase of research will improve the FPGA architectures to better support FDT computing. This will produce near term benefits for conventional FPGA applications.

The remaining phases develop significant new capabilities for fault tolerance at the chip and system level. Phase three improves dynamic reconfiguration, which currently exists in FPGAs in only limited form. Advancement is necessary to fully

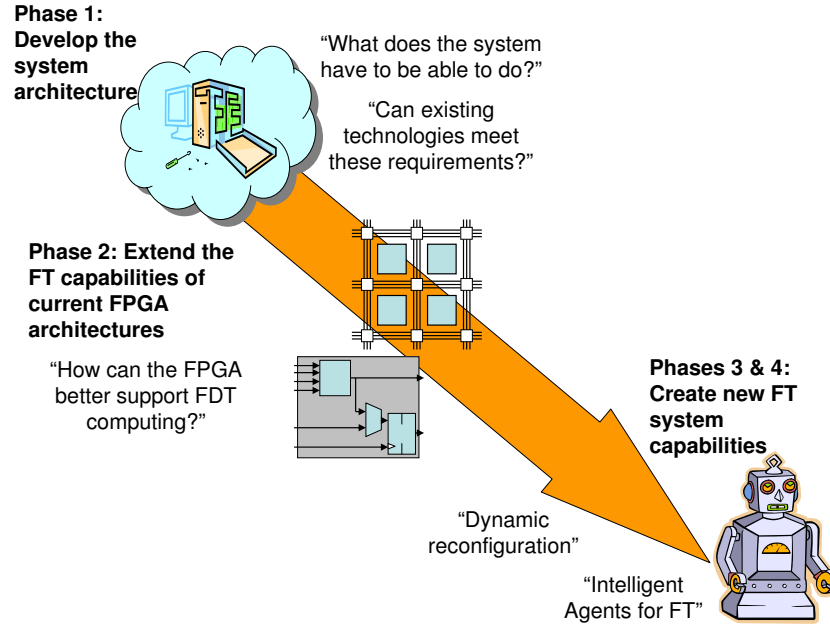


Figure B.2: Many enabling technologies will be combined to create a system architecture for a fault and defect tolerant computing.

achieve the fault tolerance benefits of reconfiguration. Phase four examines the use of intelligent and mobile agents on the FDT processor to improve fault detection and recovery and ultimately create a self-diagnosing and healing processor.

This dissertation addresses the first phase:

Develop the FDT System Architecture. Develop the system architecture for a FDT computer. Define a concept of operation, identify required capabilities and how they will work together to achieve system level fault tolerance. Identify limitations in modern FPGAs that must be improved to support the required FT capabilities.

The remainder of this chapter expands on the first research phase, outlining the applicable background as examined in Chapter II, listing three specific goals, planned contributions, and demonstration criteria to show the goals have been achieved. The methodology for achieving these goals is the subject of the next chapter. Finally, the overall group research plan includes the three follow-on research phases.

B.1.1 Current Research Focus and Shortcomings. Most of the research relevant to this area has addressed the following issues:

- CMOS scaling problems.
- Emerging device technologies to replace CMOS.
- Memory and logic devices constructed from emerging technologies.
- Computing architectures using emerging technologies.
- Fault tolerant FPGA-based architectures for aerospace applications.

To date, little work has been done on developing the system architecture of a fault tolerant computer using these emerging technologies. Most of the recent research implements fault tolerance techniques at the chip level or below and does not address support at the operating system level or higher. While the reconfigurable architecture of a FDT computer will likely resemble a modern FPGA, modifications are needed to support fault detection and recovery, as well as to target emerging device types for which the SRAM-based LUT architecture may be inefficient. These problems are the focus of this research area.

This appendix describes the three follow-on phases in the long term AFIT research effort in fault tolerant computing. These phases will expand upon the initial system architecture developed in the first research phase, providing enabling technologies for the fault and defect tolerant computer architecture.

The four phases of study for the fault and defect tolerant computing group are:

Develop the FDT System Architecture. Develop the system architecture for a FDT computer. Define a concept of operation, and identify required capabilities and how they will work together to achieve system level fault tolerance. Identify limitations in modern FPGAs that must be improved to support the required FT capabilities.

Improve FPGA Architectures. Improve the fault tolerance of modern FPGAs for use in aerospace applications.

Enable Dynamic Reconfiguration. Improve tools and architectures supporting large-scale dynamic reconfiguration to improve reconfigurability-based fault tolerance.

Enable Intelligent Agents. Expanding on the concept of roving testers implemented on a small scale, develop a capable intelligent mobile agent to test the PLD and relocate logic.

The four research phases address the problem of reliable computing at different levels of abstraction. Figure B.3 shows the relations of the four areas to each other. The higher levels of abstraction rely on capabilities provided by the lower levels. Since FDT computers will incorporate reconfigurability as a primary fault tolerance strategy, modern FPGA architectures provide a starting point to evolve FDT programmable architectures. Dynamic runtime reconfiguration must be improved at both the hardware level and in the design tools. At the next level of abstraction, the ability of the system to locate faults during operation must be improved. As part of this research, the concept of intelligent agents is adapted to the FDT computer for use in fault detection and diagnosis on the programmable array. Finally, at the top level, the overall FDT system architecture combines the capabilities into a coherent architecture that can compute reliably.

B.2 Phase 2: Improve Fault Tolerance in Current FPGAs

B.2.1 Applicable Background.

B.2.1.1 Literature Review. The following areas provide a background of the problem:

- Classical fault and defect tolerance techniques,
- Fault and defect tolerance capabilities of conventional FPGAs, and
- Partial reconfiguration capabilities of conventional FPGAs.

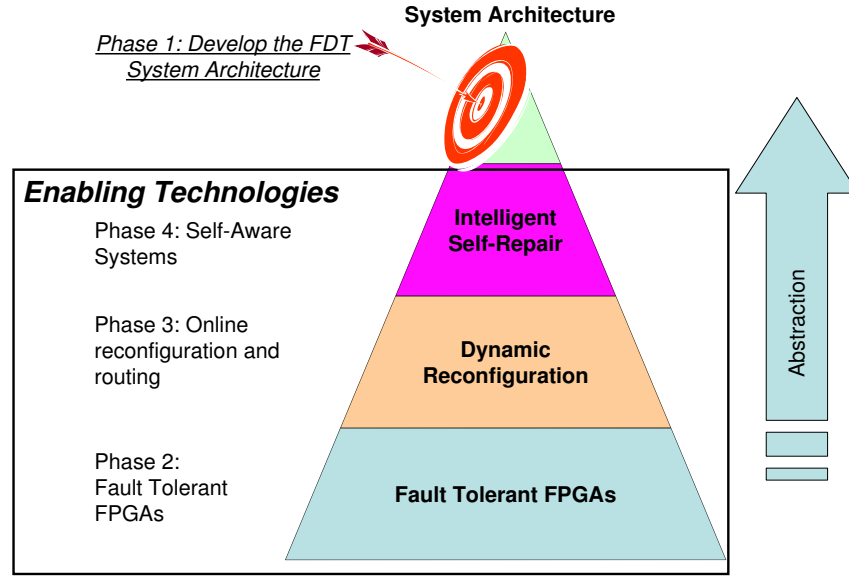


Figure B.3: Layers of Abstraction. Technologies enabling the construction of a FDT computer build upon one another.

B.2.1.2 Current Research Focus and Shortcomings. Most of the research relevant to this area addresses the following issues:

- General FPGA testing.
- Online FPGA fault detection and diagnosis.
- Minimizing the number of test configurations.
- SEU effects on FPGAs.
- Radiation effects on FPGAs.
- Fault tolerant FPGA designs (e.g., CLB, interconnect fabric, configuration memories)

While work has been done for FPGAs, fault detection and isolation to a specific resource remains difficult and time consuming. Improved methods need to be developed to rapidly detect and diagnose faults in large FPGAs.

Current FPGA architectures incorporate little fault tolerance capability beyond large amounts of configurable resources (i.e., CLBs and interconnection resources).

Current architectures do not attempt to increase the reliability of individual CLBs due to the expense of the redundant hardware and associated power consumption. While acceptable for terrestrial FPGAs, this approach increases susceptibility to single event upsets (SEUs) and other radiation effects facing aerospace users. In the future, these problems will become common in ground-based applications as well. The fault tolerance capabilities of FPGAs should be improved while minimizing the area and power impact of redundant hardware.

Runtime reconfiguration is a key capability for fault and defect tolerant computing. Current FPGAs provide limited partial reconfiguration capability. Improvements to the architectures will allow future FPGAs to support local reconfiguration done without support of the router, as well as large-scale reconfiguration under control of the router (either on-chip or running on an external processor).

B.2.2 Questions Addressed.

- How can FPGA fault detection of diagnosis be improved?
- How can SEUs in the configuration memory be detected?
- How can permanent faults in the FPGA resources be detected?
- How can the tester differentiate between SEUs and permanent faults?
- How can the fault location and impact be determined?
- How can partially functional resources be re-used?
- What architectural changes can be made to improve fault detection and diagnosis?
- What architectural changes can be made to improve fault masking and recovery?
- What is the power and hardware overhead incurred by these improvements?
- How can the benefits and disadvantages associated with making these improvements be quantified to allow design decisions to be made?

B.2.3 General Goals. The goal of this phase is to improve the fault tolerance capabilities of conventional FPGAs. From this major goal, three minor goals are defined:

1. Develop improved online FPGA fault detection and fault diagnosis methodologies that require fewer reconfigurations than current techniques such as Roving STARS.
2. Develop a modified FPGA architecture with better fault masking and fault recovery performance (when compared to standard FPGAs) against permanent faults and soft errors occurring in CLBs, SMs, and interconnect.
3. Develop a modified FPGA architecture with better fault detection, masking and fault recovery performance (when compared to standard FPGAs using config readback and scrubbing) against SEUs in CMCs and LUT memory.

B.2.4 Research Contributions.

- Improved FPGA test methods to detect and diagnose faults, potentially involving changes to the FPGA architecture.
- Improved FPGA architectures to support fault detection and diagnosis of faulty resources.
 - Develop a FPGA architecture that can localize faults to a specific CLB, switch matrix, or interconnect resource, with little or no impact on the application circuit.
 - Design a FPGA architecture that detects and isolates single event upsets in the configuration memory cells at the local level.
 - Design a FPGA CLB architecture that allows re-use of partially functional resources.
- Increase the reliability of conventional FPGAs in the presence of soft and hard errors with minimum additional hardware resources.

- Develop CLB structures to improve fault tolerance while requiring no additional routing resources. Smaller CMOS processes, as well as emerging device technologies, will provide additional transistors to the FPGA designer. Rather than being limited by transistor size, future FPGAs are limited by the size of interconnect resources.
- Develop methods to relocate application CLBs from one location to another, without support from a router.
- Develop FPGA configuration memory structures that operate more reliably in the presence of single event upsets (i.e., bit flips).
- Develop improved HW resources devoted to testing.
- Determine how conventional FPGAs can be made to operate reliably in the presence of soft and hard errors with minimum additional power consumption
 - Determine how redundancy techniques can be incorporated into FPGA structures while minimizing additional power consumption.
- Develop reliability models for FPGAs implementing FT/DT techniques.

B.2.5 Demonstration.

- Implement application circuits in VHDL, demonstrating operation of improved test methods.
- Implement new FPGA design in VHDL to demonstrate operation and improved reliability.
 - Demonstrate normal configuration and operation.
 - Demonstrate reliable operation in the presence of induced errors.
 - Demonstrate ability to isolate error to a CLB or routing resource.
 - Compare reliability predictions to simulation results.
- Implement portions of new CLB or configuration memory cell designs in SPICE to demonstrate power requirements.

B.3 Phase 3: Dynamic Reconfiguration

B.3.1 Applicable Background.

B.3.1.1 Literature Review. The following areas provide background on the problem:

- Current FPGA routing algorithms,
- On-chip routing research, and
- Dynamic partial reconfiguration capabilities of current FPGAs.

B.3.1.2 Current Research Focus and Shortcomings. Most of the research relevant to this area addresses the following issues:

- Improving the efficiency of conventional routers.
- Reducing the memory and time requirements for conventional routers.
- Performing the routing function on the FPGA as a method for device independent application distribution in mixed hardware/software applications.

Modern FPGA routers require large amounts of memory and CPU resources and are difficult to implement during application runtime. The Riverside On-Chip Router (ROCR) proposes “Just-In-Time” compilation as a method of distributing a mixed hardware/software applications to different hardware types as a single device-independent specification. The ROCR is smaller than a conventional router, but improvements must be made to implement it on a FDT processor. Current on-chip routers are intended to run on a fixed core CPU on the FPGA, not on the FPGA fabric itself. The on-chip router is not intended for fault tolerance applications, and assumes the CPU is fault-free.

To maximize the benefit of reconfiguration as a fault tolerance technique, run-time routing must be possible, both to maximize the re-use of functional resources and performance of the application circuit. If implemented on the reconfigurable mesh

itself, the runtime router must be small, fast, and efficient. It must be able to tolerate faults within the hardware implementing the router.

B.3.2 Questions Addressed.

- How can runtime routing be used to increase the fault tolerance benefits of reconfiguration?
- What are the minimum resource requirements to perform runtime routing?
- Is it feasible to perform runtime routing on the FPGA itself?
- What would be the performance impact to perform runtime routing?
- Can runtime routing be handled by the chip without the knowledge or involvement of the application circuit?
- How can runtime routing be performed reliably on an architecture that can suffer operational faults and soft errors?

B.3.3 General Goals. The goal in this area is to increase the benefit of reconfiguration as a fault tolerance technique in FPGA-based systems by enabling routing to be done during operation of the application system. Runtime routing allows large-scale reconfiguration of the application circuit on a FPGA with faulty resources, maximizing re-use of fault-free resource and improving overall system reliability.

From this major goal, three minor goals are defined:

1. Determine the minimum resource requirements to do On-Chip Routing (OCR) on the FDT mesh (i.e., implement the router on the programmable logic array rather than a fixed core CPU).
2. Implement an OCR that operates without the knowledge of the application circuit running on the FDT mesh.
3. Improve the efficiency of the OCR so that it operates with *minimum* impact on the application circuit running on the FDT mesh.

This phase will require a thorough understanding of FPGA routers and architectures. Significant background work will need to be done before the AFIT group is ready to make significant progress in this area.

B.3.4 Research Contributions.

- Demonstrate the feasibility of runtime routing on a programmable logic device.
- Predict the time and resource requirements to implement dynamic routing.
- Extend the dynamic reconfiguration capabilities of the proposed FDT design.
- Determine what extensions to the FDT node design must be made to implement dynamic routing (e.g., multi-context configuration bits, extra interconnect, etc).

B.3.5 Demonstration.

- Implement a runtime router on a FPGA-based circuit.
- Develop a VHDL model to demonstrate dynamic routing on the FDT chip.
- Demonstrate impact of OCR on operating application circuit.
- Measure the hardware resource impacts and time impact to the application circuit.

B.4 Phase 4: Intelligent Agents

B.4.1 Applicable Background.

B.4.1.1 Literature Review. The following areas provide background on the problem:

- Use of intelligent or mobile agents in fault and defect tolerant computing, and
- System-level fault detection and recovery.

B.4.1.2 Current Research and Shortcomings. Relatively little research has been directed at configuration shifting as a method of FPGA testing. As discussed in Chapter II, simple relocatable test blocks have been proposed, but they operate under the direction of an external processor which controls operation and configuration. Mobile or intelligent agents, a concept widely used in computer science, may be adapted for use in fault detection and recovery on the FPGA. This idea has not yet been explored.

B.4.2 Questions Addressed.

- Can the concept of mobile agents be adapted to fault tolerant computing to improve fault detection, diagnosis, and recovery?
- Can testing of the FPGA or FDT mesh be done without a reliable core?
- Can the tester relocate itself on the configurable mesh to test the entire device?
- How could this be done without external control?
- How would the tester relocate application logic to recover from a fault?

B.4.3 General Goals. The major goal of this research phase is to develop an intelligent mobile agent implemented in hardware on the programmable logic mesh to perform fault detection and fault recovery tasks in parallel with normal application operation. From this major goal, three minor goals are defined:

1. Develop a HW intelligent agent that can perform fault detection and diagnosis without the need for a reliable fixed core.
2. Develop a HW intelligent agent capable of relocating application logic (potentially across long distances on the FDT mesh) to handle faults.
3. Develop a HW intelligent agent that can relocate itself on the FDT mesh.

B.4.4 Research Contributions.

- Determine how the intelligent agent can verify and ensure its own health prior to start of testing.
- Determine a viable implementation of a system-wide fault recovery agent.
 - Determine whether the best implementation is a single chip-wide agent or multiple local agents. Determine the level of support required by the operating system.
 - Develop a mobile agent design capable of implementing these features.
 - Determine the required characteristics, size and performance requirements.
- Determine modifications to the FDT mesh to support mobile agent operation and relocation.

B.4.5 Demonstration.

- Implement intelligent agent in VHDL and C to control fault detection, diagnosis, and repair/reconfiguration on the FDT mesh.
- Demonstrate key abilities of the intelligent agent:
 - Reconfigures user logic to repair defects,
 - Relocates itself, and
 - Verifies its own health and correctness.

B.5 Summary

The four research phases and the associated goals are intended to form the basis of a new multi-year research group at AFIT. The four major research phases are summarized in Table B.1. This appendix creates a plan for the overall research program, and proposes an initial course of study addressing the first phase.

Table B.1: The four research phases of the AFIT Fault and Defect Tolerant Computing group.

	Phase 1	Phase 2	Phase 3	Phase 4
Major Area	FDT System Architecture	Current FPGA Architectures	Dynamic Reconfiguration	Intelligent Agents for FDT Computing
Research Focus	Develop the system architecture for a FDT computer	Improve the FT of conventional FPGAs	Increase benefit of re-configuration by enabling runtime routing in the FDT computer	Develop an intelligent agent to perform fault detection and recovery in the FDT mesh.
Goal 1	Develop the system architecture for a FDT computer, proposes CONOPS, ID required capabilities	Develop improved methods to detect and diagnose faults, minimizing overhead (HW, routing, time, and power)	Determine the minimum resource requirements to do OCR on the FDT mesh	Develop a HW intelligent agent that: Does not rely on a reliable fixed core
Goal 2	Design a FDT node/routing architecture capable of supporting required functions	Improve the FPGA architecture's ability to mask, detect, and recover from operational permanent faults	Design an OCR that operates without the knowledge of the application circuit	Can relocate logic to re-use fault-free resources
Goal 3	Develop techniques to map FDT nodes onto emerging technologies	Improve the FPGA architecture's ability to detect and recover from SEUs	Improve the efficiency of the OCR so that it operates with minimum impact on the application circuit	Can relocate itself on the FDT mesh

Bibliography

- ACD⁺02. M. Alderighi, F. Casini, S. D'Angelo, D. Salvi, and G.R. Sechi. A fault-tolerant FPGA-based multi-stage interconnection network for space applications. In *Proceedings of the First IEEE Int'l Workshop on Electronic Design, Test and Applications*, pages 302–306, 2002.
- ACKH01. K. Asano, Y.-K. Choi, T.-J. King, and C. Hu. Patterning sub-30nm MOSFET gate with I-line lithography. *IEEE Transactions on Electron Devices*, 48:1004–1006, May 2001.
- Adv05. Advanced Micro Devices, inc. *AMD Athlon 64 Functional Data Sheet, 939 pin package*, rev 3.03 edition, May 2005.
- AES01. M. Abramovici, J.M. Emmert, and C.E. Stroud. Roving STARs: An integrated approach to on-line testing, diagnosis, and fault tolerance for FPGAs in adaptive computing systems. In *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware*, pages 73–92, 2001.
- AL81. T. Anderson and P.A. Lee. *Fault Tolerance: Principles and Practice*. Prentice Hall, 1981.
- All90. Arnold O. Allen. *Probability, Statistics, and Queueing Theory*. Computer Science and Scientific Computing. Academic Press, Boston, 2nd edition, 1990.
- AMZE04. G. Asadi, S.G. Miremadi, H.R. Zarandi, and A. Ejlali. Evaluation of fault-tolerant designs implemented on SRAM-based FPGAs. In *Proceedings of the 10th IEEE Pacific Rim International Symposium on Dependable Computing*, pages 327–332, 2004.
- AOK⁺00. I. Amlani, A.O. Orlov, R.K. Kummamuru, G.H. Bernstein, C.S. Lent, and G.L. Snider. Experimental demonstration of a leadless quantum-dot cellular automata. *Applied Physics Letters*, 77(5):738–740, 2000.
- AOT⁺99. Islamshah Amlani, Alexei O. Orlov, Geza Toth, Gary H. Bernstein, Craig S. Lent, and Gregory L. Snider. Digital logic gate using quantum-cellular automata. *Science*, 284:289–291, 1999.
- AS93. Peter M. Athanas and Harvey F. Silverman. Processor reconfiguration through instruction set metamorphosis. *IEEE Computer*, 26(3):11–18, March 1993.
- ASE04. M. Abramovici, C.E. Stroud, and J.M. Emmert. Online BIST and BIST-based diagnosis of FPGA logic blocks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(12):1284–1294, 2004.

- ASH⁺99. Miron Abramovici, Charles Stroud, Carter Hamilton, Sajitha Wijesuriya, and Vinay Verma. Using Roving STARs for on-line testing and diagnosis of FPGAs in fault-tolerant applications. In *Proceedings of the IEEE International Test Conference*, pages 973–982, 1999.
- ASSE00. M. Abramovici, C. Stroud, B. Skaggs, and J. Emmert. Improving on-line BIST-based diagnosis for Roving STARs. In *Proceedings of the 6th IEEE International On-Line Testing Workshop*, pages 31–39, 2000.
- BGM04. M.A. Breuer, S.K. Gupta, and T.M. Mak. Defect and error tolerance in the presence of massive numbers of defects. *IEEE Design & Test of Computers*, 21(3):216–227, 2004.
- BH98. P. Bellows and B. Hutchings. JHDL-an HDL for reconfigurable systems. In *Proceedings of the 1998 IEEE Symposium on FPGAs for Custom Computing Machines*, pages 175–184, Los Alamitos, CA, 1998. IEEE Computer Society Press.
- BMH00. I. Boyarinov, I. Martin, and B. Honary. High-speed decoding of extended Golay code. *IEE Proceedings on Communications*, 147(6):333–336, 2000.
- Boh03. Mark Bohr. Intel’s 90nm logic technology using strained silicon. Press Briefing, December 2003.
- Bou03. G. Bourianoff. The future of nanocomputing. *IEEE Computer*, 36(8):44–53, August 2003.
- BQA03. V. Beiu, J.M. Quintana, and M.J. Avedillo. VLSI implementations of threshold logic-a comprehensive survey. *IEEE Transactions on Neural Networks*, 14(5):1217–1243, 2003.
- BRM99. V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs*. Kluwer Academic Publishers, 1999.
- BRM03. V. Betz, J. Rose, and A. Marquardt. VPR and T-VPack: Versatile packing, placement, and routing for FPGAs. Online: <http://www.eecg.toronto.edu/~vaughn/vpr/vpr.html>, 2003.
- BRSV04. P. Bernardi, M.S. Reorda, L. Sterpone, and M. Violante. On the evaluation of SEU sensitiveness in SRAM-based FPGAs. In *Proceedings of the 10th Int’l Online Testing Symposium (IOLTS04)*, pages 115–120, 2004.
- BRV89. P. Bertin, D. Roncin, and J. Vuillemin. Introduction to programmable active memories. Technical Report 3, DEC Paris Research Laboratory, 1989.
- BRV92. S. Brown, J. Rose, and Z.G. Vranesic. A detailed router for field-programmable gate arrays. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(5):620–628, 1992.

- BS04a. D. Bhaduri and S. Shukla. Tools and techniques for evaluating reliability of defect-tolerant nano architectures. In *Proceedings of the 2004 IEEE International Joint Conference on Neural Networks*, volume 4, pages 2641–2646 vol.4, 2004.
- BS04b. D. Bhaduri and S.K. Shukla. Reliability evaluation of von Neumann multiplexing based defect-tolerant majority circuits. In *4th IEEE Conference on Nanotechnology*, pages 599–601, 2004.
- CAA01. P.G. Collins, M. Arnold, and P. Avouris. Engineering carbon nanotubes and nanotube circuits using electrical breakdown. *Science*, 292:706, 2001.
- Car01. Carl Carmichael. *Triple Modular Redundancy Design Techniques for Virtex FPGAs, v1.0*. Xilinx, inc., Nov 2001.
- CCH⁺03. L. Chang, Yang-kyu Choi, D. Ha, P. Ranade, Shiyong Xiong, J. Bokor, Chenming Hu, and T.J. King. Extremely scaled silicon nano-CMOS devices. *Proceedings of the IEEE*, 91(11):1860–1873, 2003.
- CFBC99. Carl Carmichael, Earl Fuller, Phil Blain, and Michael Caffrey. SEU mitigation techniques for Virtex FPGAs in space applications. In *Proceedings of the 1999 Conference on Military Applications of Programmable Logic Devices (MAPLD1999)*, 1999.
- Cha94. Pak K. Chan. *Digital Design Using Field Programmable Gate Arrays*. PTR Prentice Hall, 1994.
- Cla98. D. Clark. Teramac: Pointing the way to real-world nanotechnology. *IEEE Computational Science and Engineering*, 5(3):88–90, 1998.
- CLC⁺02. K. Compton, Zhiyuan Li, J. Cooley, S. Knol, and S. Hauck. Configuration relocation and defragmentation for run-time reconfigurable computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 10(3):209–220, 2002.
- CLM⁺03. Gian Carlo Cardarilli, Alessandro Leandri, Panfilo Marinucci, Marco Ottavi, Salvatore Pontarelli, Marco Re, and Adelio Salsano. Design of a fault tolerant solid state mass memory. *IEEE Transactions on Reliability*, 52(4):476–491, Dec 2003.
- CNV96. T. Calin, M. Nicolaidis, and R. Velazco. Upset hardened memory design for submicron CMOS technology. *IEEE Transactions on Nuclear Science*, 43(6):2874–2878, 1996.
- Com99. K. Compton. Programming architectures for run-time reconfigurable systems. Master’s thesis, Northwestern University, 1999.
- Com00. R. Compano. Technology roadmap for nanoelectronics, 2nd. ed. Technical report, Microelectronics Advanced Research Initiative (MELARI NANO), Nov 2000.

- Cou06. David Coursey. Microsoft approaches secure, reliable Vista. eWeek, Feb 2006.
- CPL⁺03. M. Choi, N. Park, F. Lombardi, Y.B. Kim, and V. Piuri. Optimal spare utilization in repairable and reliable memory cores. In *Records of the 2003 International Workshop on Memory Technology, Design and Testing*, pages 64–71, 2003.
- CS97. P.K. Chan and M.D.F. Schlag. Acceleration of an FPGA router. In *Proceedings of the 5th Annual IEEE Symposium on FPGAs for Custom Computing Machines (FCCM97)*, pages 175–181, 1997.
- CTS95. Steve Casselman, Michael Thornburg, and John Schewel. Hardware object programming on the EVC1: a reconfigurable computer. In *Proceedings of the SPIE 2607: FPGAs for Fast Board Development and Reconfigurable Computing*, Billingham, WA, 1995.
- Cun90. J. A. Cunningham. The use and evaluation of yield models in integrated circuit manufacturing. *IEEE Transactions on Semiconductor Manufacturing*, 3:60–71, 1990.
- CVR⁺03. M. Ceschia, M. Violante, M.S. Reorda, A. Paccagnella, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, and A. Candelori. Identification and classification of single-event upsets in the configuration memory of SRAM-based FPGAs. *IEEE Transactions on Nuclear Science*, 50(6):2088–2094, 2003.
- CWB⁺99. C.P. Collier, E.W. Wong, M. Belohradsky, F.M. Raymo, J.F. Stoddart, P.J. Kuekes, R.S. Williams, and J.R. Heath. Electronically configurable molecular-based logic gates. *Science*, 285:391–394, Jul 1999.
- DeJ98. Ruth DeJule. New fab construction. *Semiconductor International*, 21(1):81–86, Jan 1998.
- DI99. A. Doumar and H. Ito. Testing the logic cells and interconnect resources for FPGAs. In *Proceedings of the Eighth Asian Test Symposium (ATS '99)*, pages 369–374, 1999.
- DI00. A. Doumar and H. Ito. Testing approach within FPGA-based fault tolerant systems. In *Proceedings of the Ninth Asian Test Symposium (ATS 2000)*, pages 411–416, Dec 2000.
- DI01. A. Doumar and H. Ito. FPGAs and fault tolerance. In *Proceedings of the 13th Intl Conference on Microelectronics*, pages 222–225, 2001.
- DI03. A. Doumar and H. Ito. Detecting, diagnosing, and tolerating faults in SRAM-based field programmable gate arrays: a survey. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 11(3):386–405, 2003.

- DK03. T.J. Dysart and P.M. Kogge. Strategy and prototype tool for doing fault modeling in a nano-technology. In *Proceedings of the 2003 Third IEEE Conference on Nanotechnology (IEEE-NANO 2003)*, volume 1, pages 356–359 vol.2, 2003.
- DKI99. A. Doumar, S. Kaneko, and H. Ito. Defect and fault tolerance FPGAs by shifting the configuration data. In *Proceedings of the International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT '99)*, pages 377–385, 1999.
- dLKNH⁺04. F.G. de Lima Kastensmidt, G. Neuberger, R.F. Hentschke, L. Carro, and R. Reis. Designing fault-tolerant techniques for SRAM-based FPGAs. *IEEE Design & Test of Computers*, 21(6):552–562, 2004.
- DMP⁺98. S. D’Angelo, C. Metra, S. Pastore, A. Pogutz, and G.R. Sechi. Fault-tolerant voting mechanism and recovery scheme for TMR FPGA-based systems. In *Proceedings of the 1998 IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT98)*, pages 233–240, Nov 1998.
- DPTV04. Chris Dwyer, John Poulton, Russell Taylor, and Leandra Vicci. DNA self-assembled parallel computer architectures. *Nanotechnology*, 15:1688–1694, 2004.
- DST99. D. Deshpande, A.K. Somani, and A. Tyagi. Configuration caching vs. data caching for striped FPGAs. In *Proceedings of the 1999 ACM/SIGDA Int’l Symposium on FPGAs*, pages 206–214, 1999.
- DVP⁺04. C. Dwyer, L. Vicci, J. Poulton, D. Erie, R. Superfine, S. Washburn, and II Taylor, R.M. The design of DNA self-assembled computing circuitry. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(11):1214–1220, 2004.
- EB98. J.M. Emmert and D. Bhatia. Incremental routing in FPGAs. In *Proceedings of the Eleventh Annual IEEE International ASIC Conference*, pages 217–221, 1998.
- Els03. Khaled Elshafey. Embedding fault tolerance via reconfiguration in configurable systems. In *Proceedings of the ICM*, pages 370–373, December 2003.
- EMHB95. C. Ebeling, L. McMurchie, S.A. Hauck, and S. Burns. Placement and routing tools for the Triptych FPGA. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 3(4):473–482, 1995.
- EP98. W. Evans and N. Pippenger. On the maximum tolerable noise for reliable computation by formulas. *IEEE Transactions on Information Theory*, 44(3):1299–1305, May 1998.

- FNS01. M. Forshaw, N. Nikolic, and A. Sadek. EC ANSWERS project, third year report (Melari 28667). Technical report, University College London, 2001.
- FT01. A. Fijany and B.N. Toomarian. New design for quantum dot cellular automata to obtain fault tolerant logic gates. *Journal of Nanoparticle Research*, 3:27–37, 2001.
- GAA⁺03. T. Ghani, M. Armstrong, C. Auth, M. Bost, P. Charvat, G. Glass, T. Hoffmann, K. Johnson, C. Kenyon, J. Klaus, B. McIntyre, K. Mistry, A. Murthy, J. Sandford, M. Silberstein, S. Sivakumar, P. Smith, K. Zawadzki, S. Thompson, and M. Bohr. A 90nm high volume manufacturing logic technology featuring novel 45nm gate length strained silicon CMOS transistors. In *Technical Digest of the IEEE International Electron Devices Meeting (IEDM '03)*, pages 11.6.1–11.6.3, 2003.
- Gal95. David Galloway. The Transmogripher C hardware description language and compiler for FPGAs. In *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines (FCCM95)*, April 1995.
- GB95. E. Olusegun George and Dale Bowman. A full likelihood procedure for analysing exchangeable binary data. *Biometrics*, 51:512–523, June 1995.
- Gep02. L. Geppert. The amazing vanishing transistor act. *IEEE Spectrum*, 39(10):28–33, 2002.
- GMI⁺99. M. Governale, M. Macucci, G. Iannaccone, C. Ungarelli, and J. Martorell. Modeling and manufacturability assessment of bistable quantum-dot cells. *Journal of Applied Physics*, 85(5):2962–2971, 1999.
- GP98. K. Goser and C. Pacha. System and circuit aspects of nanoelectronics. In *Proceedings of the 24th European Solid-State Circuits Conference (ESSCIRC '98)*, pages 18–29, 1998.
- Hau98. Scott Hauck. Configuration pre-fetch for single context reconfigurable coprocessors. In *Proceedings of the 1998 ACM/SIGDA Int'l Symposium on FPGAs*, pages 65–74, 1998.
- HBH⁺99. B. Hutchings, P. Bellows, J. Hawkins, S. Hemmert, B. Nelson, and M. Rytting. A CAD suite for high-performance FPGA design. In *Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM '99)*, pages 12–24, 1999.
- HBZB02. J.A. Hutchby, G.I. Bourianoff, V.V. Zhirnov, and J.E. Brewer. Extending the road beyond CMOS. *IEEE Circuits and Devices Magazine*, 18(2):28–41, 2002.

- HD98. F. Hanchek and S. Dutt. Methodologies for tolerating cell and interconnect faults in FPGAs. *IEEE Transactions on Computers*, 47(1):15–33, 1998.
- HJ02. Jie Han and Pieter Jonker. A system architecture solution for unreliable nanoelectronic devices. *IEEE Transactions on Nanotechnology*, 1(4):201–208, December 2002.
- HJ03. Jie Han and Pieter Jonker. A defect- and fault-tolerant architecture for nanocomputers. *Nanotechnology*, 14(2):224–230, 2003.
- HKSW98. James R. Heath, Philip J. Kuekes, Gregory S. Snider, and R. Stanley Williams. A defect-tolerant computer architecture: Opportunities for nanotechnology. *Science*, 280:1716–1721, 1998.
- HL96. W.K. Huang and F. Lombardi. An approach for testing programmable/-configurable field programmable gate arrays. In *Proceedings of 14th VLSI Test Symposium (VTS96)*, pages 450–455, 1996.
- HL01. K. Hennessy and C.S. Lent. Clocking of molecular quantum-dot cellular automata. *Journal of Vacuum Science and Technology*, 19(5):1752–1755, 2001.
- HNE⁺02. J.L. Hoyt, H.M. Nayfeh, S. Eguchi, I. Aberg, G. Xia, T. Drake, E.A. Fitzgerald, and D.A. Antoniadis. Strained silicon MOSFET technology. In *Technical Digest of the 2002 International Electron Devices Meeting (IEDM '02)*, pages 23–26, 2002.
- HPS75. D. Hampel, K.J. Prost, and N.R. Scheinberg. Threshold logic using complementary MOS devices. U.S. Patent Number 3,900,742, August 1975.
- HSA94. Andrew Holmes-Siedle and Len Adams. *Handbook of Radiation Effects*. Oxford Science Publications, 1994.
- HSN⁺93. F. Hatori, T. Sakurai, K. Nogami, K. Sawada, M. Takahashi, M. Ichida, M. Uchida, I. Yoshii, Y. Kawahara, T. Hibi, Y. Saeki, H. Muroga, A. Tanaka, and K. Kanzaki. Introducing redundancy in field programmable gate arrays. In *Proceedings of the IEEE 1993 Custom Integrated Circuits Conference*, pages 7.1.1–7.1.4, 1993.
- HTA94. N.J. Howard, A.M. Tyrrell, and N.M. Allinson. The yield enhancement of field-programmable gate arrays. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(1):115–123, 1994.
- HTL04a. J. Huang, M.B. Tahoori, and F. Lombardi. Fault tolerance of programmable switch blocks. In *Proceedings of the 2004 Design, Automation, and Test in Europe Conference (DATE04)*, 2004.

- HTL04b. J. Huang, M.B. Tahoori, and F. Lombardi. Probabilistic analysis of fault tolerance of FPGA switch block array. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pages 145–152, Apr 2004.
- HTL04c. Jing Huang, M.B. Tahoori, and F. Lombardi. Routability and fault tolerance of FPGA interconnect architectures. In *Proceedings of the 2004 International Test Conference*, pages 479–488, 2004.
- HWLB02. Q. Hang, Y. Wang, M. Lieberman, and G.H. Bernstein. Molecular patterning through high-resolution polymethylmethacrylate masks. *Applied Physics Letters*, 80(22):4220–4222, 2002.
- IS95. Christian Iseli and Eduardo Sanchez. A C++ compiler for FPGA custom execution units synthesis. In *Proceedings of the IEEE Symposium on Custom Computing Machines (FCCM95)*, April 1995.
- JLG⁺03. J. Jiao, G.L. Long, F. Grandjean, A.M. Beatty, and T.P. Fehiner. Building blocks for the molecular expression of qca, isolation and characterization of a covalently bounded square array of two ferrocenium and two ferrocene complexes. *Journal of the American Chemical Society*, 125(25):7522–7523, 2003.
- KH04. T. Karnik and P. Hazucha. Characterization of soft errors caused by single event upsets in CMOS processes. *IEEE Transactions on Dependable and Secure Computing*, 1(2):128–143, 2004.
- KI94. J.L. Kelly and P.A. Ivey. Defect tolerant SRAM-based FPGAs. In *Proceedings of the IEEE International Conference on Computer Design VLSI in Computers and Processors (ICCD '94)*, pages 479–482, 1994.
- KK97. I. Koren and Z. Koren. Analysis of a hybrid defect-tolerance scheme for high-density memory ICs. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT97)*, pages 166–174, 1997.
- KK98. I. Koren and Z. Koren. Defect tolerance in VLSI circuits: techniques and yield analysis. *Proceedings of the IEEE*, 86(9):1819–1838, 1998.
- KM02. Eric Keller and Scott McMillan. An FPGA wire database for run-time routers. In *Proceedings on the 2002 International Conference on Military Applications of Programmable Logic Devices (MAPLD02)*, 2002.
- Kor89. Israel Koren. *Defect and Fault Tolerance in VLSI Systems*, volume 1. Plenum Press, 1989.
- KW02. P.J. Kuekes and R.S. Williams. Defect-tolerant molecular electronics. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2002)*, volume 2, pages II–42–II–44 vol.2, 2002.

- KZJS00. D. Keymeulen, R.S. Zebulum, Y. Jin, and A. Stoica. Fault-tolerant evolvable hardware using field-programmable transistor arrays. *IEEE Transactions on Reliability*, 49(3):305–316, 2000.
- LC67. P.M. Lewis and C.L. Coates. *Threshold Logic*. John Wiley & Sons, 1967.
- LCC⁺04. C. Le, S. Chan, F. Cheng, W. Fang, M. Fischman, S. Hensley, R. Johnson, M. Jourdan, M. Marina, B. Parham, F. Rogez, P. Rosen, B. Shah, and S. Taft. Onboard FPGA-based SAR processing for future spaceborne systems. In *Proceedings of the IEEE Radar Conference*, pages 15–20, 2004.
- LCH00. Z. Li, K. Compton, and S. Hauck. Configuration caching management techniques for reconfigurable computing. In *Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM00)*, pages 22–36, 2000.
- LCR03. F. Lima, L. Carro, and R. Reis. Designing fault tolerant systems into SRAM-based FPGAs. In *Proceedings of the 2003 Design Automation Conference (DAC03)*, pages 650–655, 2003.
- LDJC83. Shu Lin and Jr. Daniel J. Costello. *Error Control Coding: Fundamentals and Applications*. Prentice-Hall, 1983.
- Lee61. C.Y. Lee. An algorithm for path connection and its applications. *IRE Transaction on Electronic Computing*, 10:346–365, 1961.
- Lie02. M. Lieberman. Quantum-dot cellular automata at a molecular scale. *Annals of the N.Y. Academy of Science*, 960:225–239, 2002.
- LIL03. C.S. Lent, B. Isaksen, and M. Lieberman. Molecular quantum dot cellular automata. *Journal of the American Chemical Society*, 125:1056–1063, 2003.
- LKC04. Myung-Hyun Lee, Young Kwan Kim, and Yoon-Hwa Choi. A defect-tolerant memory architecture for molecular electronics. *IEEE Transactions on Nanotechnology*, 3(1):152–157, 2004.
- LL05. INRIA Lorraine and LORIA. Multiple-precision floating point library (MPFR). Software library, Dec 2005.
- LMSL05. Jung Hoon Lee, Xialong Ma, Dmitri B. Strukov, and Konstantin K. Likharev. CMOL. In *IEEE Int’l Workshop on Design and Test of Defect Tolerant Nanoscale Architectures (NANOARCH 2005)*, pages 3.9–3.16. IEEE Computer Society, May 2005.
- LMSP98. J. Lach, W.H. Mangione-Smith, and M. Potkonjak. Low overhead fault-tolerant FPGA systems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 6(2):212–221, 1998.

- Lo93. J.-C. Lo. Fault-tolerant content addressable memory. In *Proceedings of the 1993 IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD '93)*, pages 193–196, 1993.
- Lo94. Jien-Chung Lo. A fault-tolerant associative approach to on-line memory repair. In *Proceedings of the IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems (DFT94)*, pages 168–176, 1994.
- LT94. C.S. Lent and P.D. Tougaw. Bistable saturation due to single electron charging in rings of tunnel junctions. *Journal of Applied Physics*, 75:4077, 1994.
- LT97. C.S. Lent and P.D. Tougaw. A device architecture for computing with quantum dots. *Proceedings of the IEEE*, 85:541, 1997.
- LTPB93. Craig S. Lent, P. Douglas Tougaw, Wolfgang Porod, and Gary H. Bernstein. Quantum cellular automata. *Nanotechnology*, 4:49–57, 1993.
- LVT04a. R. Lysecky, F. Vahid, and S.X.-D. Tan. Dynamic FPGA routing for just-in-time FPGA compilation. In *Proceedings of the 41st Design Automation Conference (DAC04)*, pages 954–959, 2004.
- LVT04b. Roman Lysecky, Frank Vahid, and Sheldon Tan. Dynamic FPGA routing for just-in-time compilation (Powerpoint presentation). 41st Design Automation Conference (DAC04), 2004.
- Man88. M. Morris Mano. *Computer Engineering Hardware Design*. Prentice Hall, 1988.
- MC04. Clive Maxfield and Mentor Graphics Corp. *The Design Warrior's Guide to FPGAs: Devices, Tools, and Flows*. Elsevier, 200 Wheeler Rd, Burlington, MA, 2004.
- MHL05. M. Momenzadeh, J. Huang, and F. Lombardi. Defect characterization and tolerance of QCA sequential devices and circuits. In *Proc. of the 2005 20th IEEE Int'l Symp. on Defect and Fault Tolerance in VLSI Systems (DFT05)*, 2005.
- MHS⁺04. Subhasish Mitra, W.-J. Huang, N.R. Saxena, S.-Y. Yu, and E.J. McCluskey. Reconfigurable architecture for autonomous self-repair. *IEEE Design & Test of Computers*, 21(3):228–240, 2004.
- MHTL05. M. Momenzadeh, Jing Huang, M.B. Tahoori, and F. Lombardi. On the evaluation of scaling of QCA devices in the presence of defects at manufacturing. *IEEE Transactions on Nanotechnology*, 4(6):740–743, 2005.
- MOL05. M. Momenzadeh, M. Ottavi, and F. Lombardi. Modeling QCA defects at molecular-level in combinational circuits. In *20th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2005)*, pages 208–216, 2005.

- Moo65. Gordon E. Moore. Cramming more components into integrated circuits. *Electronics*, 38(8):1–4, April 1965.
- MP00. Silvia Mueller and Wolfgang Paul. *Computer Architecture: Complexity and Correctness*. Springer-Verlag, 2000.
- MSS⁺98. R. Martel, T. Schmidt, H.R. Shea, T. Hertel, and P. Avouris. Single- and multi-wall carbon nanotube field-effect transistors. *Applied Physics Letters*, 73:2447, 1998.
- MTHL04. M. Momenzadeh, M.B. Tahoori, J. Huang, and F. Lombard. Quantum cellular automata: new defects and faults for new devices. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, pages 207–, 2004.
- MVM90. T.L. Michalka, R.C. Varshney, and J.D. Meindl. A discussion of yield modeling with defect clustering, circuitrepair, and circuit redundancy. *IEEE Transactions on Semiconductor Manufacturing*, 3(3):116–127, 1990.
- Nel90. Victor P. Nelson. Fault-tolerant computing: Fundamental concepts. *IEEE Computer*, 23:20–25, 1990.
- NF01. K. Nikolic and M. Forshaw. The relative success of nanoscale RTD, SET and EQCA devices as replacements for CMOS at the system level. In *Nanotechnology, 2001. IEEE-NANO 2001. Proceedings of the 2001 1st IEEE Conference on*, pages 272–276, 2001.
- Nik96. D. Nikolos. Yield - performance tradeoffs for VLSI processors with partially-good two-level on-chip caches. *Proceedings of the 1996 Workshop on Defect and Fault Tolerance in VLSI Systems (DFT96)*, 1996.
- NK99. M.T. Niemier and P.M. Kogge. Logic in wire: using quantum dots to implement a microprocessor. In *Proceedings of the 6th IEEE International Conference on Electronics, Circuits and Systems (ICECS '99)*, volume 3, pages 1211–1215 vol.3, 1999.
- NK01. M.T. Niemier and P.M. Kogge. Exploring and exploiting wire-level pipelining in emerging technologies. In *Proceedings of the 28th Annual International Symposium on Computer Architecture (ISCA)*, pages 166–177, 2001.
- NK04. M.T. Niemier and P.M. Kogge. The “4-diamond circui” - a minimally complex nano-scale computational building block in QCA. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 3–10, 2004.
- NPK04. Gethin Norman, David Parker, and Marta Kwiatkowska. Evaluating the reliability of defect-tolerant architectures for nanotechnology with

- probabilistic model checking. In *Proceedings of the 17th Int'l Conf. on VLSI Design (VLSID04)*, 2004.
- NRK04. M.T. Niemier, R. Ravichandran, and P.M. Kogge. Using circuits and systems-level research to drive nanotechnology. In *Proceedings of the IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD 2004)*, pages 302–309, 2004.
- NSF01. K. Nikolic, A. Sadek, and M. Forshaw. Architectures for reliable computing with unreliable nanodevices. In *Proceedings of the 2001 1st IEEE Conference on Nanotechnology (IEEE-NANO 2001)*, pages 254–259, 2001.
- NV99. D. Nikolos and H.T. Vergos. On the yield of VLSI processors with on-chip CPU cache. *IEEE Transactions on Computers*, 48(10):1138–1144, 1999.
- OVLP05. M. Ottavi, V. Vankamamidi, F. Lombardi, and S. Pontarelli. Novel memory designs for QCA implementation. In *Proceedings of the 5th IEEE Conference on Nanotechnology (NANO05)*, pages 545–548 vol. 2, 2005.
- PCL⁺02. S. Pontarelli, G.C. Cardarilli, A. Leandri, M. Ottavi, M. Re, and A. Salsano. A self-checking cell logic block for fault tolerant FPGAs. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 2002)*, volume 4, pages IV–477–IV–480 vol.4, May 2002.
- PH98. David Patterson and John Hennessy. *Computer Organization & Design: The Hardware/Software Interface*. Morgan Kaufman, 1998.
- PJS⁺01. A. R. Pease, J.O. Jeppeson, J.F. Stoddart, Y. Luo, C.P. Collier, and J.R. Heath. Switching devices based on interlocked molecules. *Accounts of Chemical Research*, 34(6):433–444, 2001.
- PNK04. Dave Parker, Gethin Norman, and Marta Kwiatkowska. *PRISM 2.1 Users' Guide*. University of Birmingham, Sep 2004.
- Raj92. Rochit Rajsuman. *Digital Hardware Testing: Transistor-Level Fault Modeling and Testing*. Artech House, Norwood, MA, 1992.
- Rat04. David Ratter. FPGAs on Mars. *XCell Journal*, 50(50):8–12, 2004.
- RB04. S. Roy and V. Beiu. Multiplexing schemes for cost-effective fault-tolerance. In *Proceedings of the 4th IEEE Conference on Nanotechnology (NANO2004)*, pages 589–592, 2004.
- RB05. S. Roy and V. Beiu. Majority multiplexing-economical redundant fault-tolerant designs for nanoarchitectures. *IEEE Transactions on Nanotechnology*, 4(4):441–451, 2005.

- RBB06. George Roelke, Rusty Baldwin, and Dursun Bulutoglu. Analytical models for the performance of von Neumann multiplexing. *accepted for publication in IEEE Transactions on Nanotechnology*, 2006.
- RBMK06a. George Roelke, Rusty O. Baldwin, Barry Mullins, and Yong C. Kim. A fault tolerant content addressable memory cache architecture for nanotechnologies. In *2nd IEEE International Workshop on Defect and Fault Tolerant Nanoscale Architectures (NANORARCH 2006)*, 2006.
- RBMK06b. George R. Roelke, Rusty O. Baldwin, Barry Mullins, and Yong C. Kim. A content addressable memory cache architecture for extremely unreliable nanotechnologies. *submitted to IEEE Transactions on Reliability*, 2006.
- Rei00. Rudiger Reischuk. Can large fanin circuits perform reliable computations in the presence of faults? *Theoretical Computer Science*, 240(2):319–335, June 2000.
- Ris02. L. Risch. The end of the CMOS roadmap- new landscape beyond. *Material Science & Engineering C*, 19:363–368, 2002.
- RN95. K. Roy and S. Nag. On routability for FPGAs under faulty conditions. *IEEE Transactions on Computers*, 44(11):1296–1305, 1995.
- Roe97. George R. Roelke. A framework for an automated compilation system for reconfigurable architectures. Masters thesis, Air Force Institute of Technology, March 1997.
- RPFZ98. M. Renovell, J.M. Portal, J. Figueras, and Y. Zorian. Testing the interconnect of RAM-based FPGAs. *IEEE Design & Test of Computers*, 15(1):45–50, 1998.
- RPFZ99. M. Renovell, J.M. Portal, J. Figueras, and Y. Zorian. Testing the configurable interconnect/logic interface of SRAM-based FPGAs. In *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE1999)*, pages 618–622, 1999.
- RWCG02. Nathan Rollins, Michael Wirthlin, Michael Caffrey, and Paul Graham. Reliability of programmable Input/Output pins in the presence of configuration upsets. In *Proceedings of the 2002 Conference on Military Applications of Programmable Logic Devices (MAPLD2002)*, 2002.
- Sem03. Semiconductor Industry Association. 2003 international technology roadmap for semiconductors (ITRS). Technical report, Semiconductor Industry Association, 2003.
- Ser03. George Sery. Approaching the one billion transistor logic product: Process and design challenges. Intel Corporation Powerpoint Presentation, Dec 2003.

- SGV⁺04. S. Srinivasan, A. Gayasen, N. Vijaykrishnan, M. Kandemir, Y. Xie, and M.J. Irwin. Improving soft-error tolerance of FPGA configuration bits. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD-2004)*, pages 107–110, 2004.
- SK92. C.H. Stapper and W.A. Klaasen. The evaluation of 16-Mbit memory chips with built-in reliability. In *Proceedings of the 30th Annual International Reliability Physics Symposium*, pages 3–7, 1992.
- SKCA96. C. Stroud, S. Konala, Ping Chen, and M. Abramovici. Built-in self-test of logic blocks in FPGAs (finally, a free lunch BIST without overhead!). In *Proceedings of 14th VLSI Test Symposium (VTS96)*, pages 387–392, 1996.
- Skl01. Bernard Sklar. *Digital Communications: Fundamentals and Applications*. Prentice Hall PTR, 2nd edition, 2001.
- SL05. D.B. Strukov and K.K. Likharev. CMOL FPGA: A reconfigurable architecture for hybrid digital circuits with two-terminal nanodevices. *Nanotechnology*, 16:888–900, April 2005.
- SLA97. C. Stroud, E. Lee, and M. Abramovici. BIST-based diagnostics of FPGA logic blocks. In *Proceedings of the 1997 Int’l Test Conference*, pages 539–547, 1997.
- SMSP97. N.R. Shnidman, W.H. Mangione-Smith, and M. Potkonjak. Fault scanner for reconfigurable logic. In *Proceedings of the Seventeenth Conference on Advanced Research in VLSI*, pages 238–255, 1997.
- SNF04. Akram S. Sadek, Konstantin Nikolic, and Michael Forshaw. Parallel information and computation with restitution for noise-tolerant nanoscale logic networks. *Nanotechnology*, 15(1):192–210, 2004.
- Sni98. G.L. Snider. A functional cell for quantum-dot cellular automata. *Solid State Electronics*, 42(7-8):1355–1359, 1998.
- SP03. A.P. Shanthi and R. Parthasarathi. Exploring FPGA structures for evolving fault tolerant hardware. In *Proceedings of the NASA/DoD Conference on Evolvable Hardware*, pages 174–181, Jul 2003.
- Spr79. Melvin D. Springer. *The Algebra of Random Variables*. John Wiley & Sons, New York, 1979.
- SSS02. F. Salice, M.G. Sami, and R. Stefanelli. Fault-tolerant CAM architectures: a design framework. In *Proceedings of the 17th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems (DFT 2002)*, pages 233–241, 2002.
- Sta93. C.H. Stapper. Improved yield models for fault-tolerant memory chips. *IEEE Transactions on Computers*, 42(7):872–881, 1993.

- Sto99. A. Stoica. Toward evolvable hardware chips: Experiments with a programmable transistor array. In *Proceedings of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems (MicroNeuro '99)*, pages 156–162, 1999.
- SWHA98. C. Stroud, S. Wijesuriya, C. Hamilton, and M. Abramovici. Built-in self-test of FPGA interconnect. In *Proceedings of the 1998 International Test Conference*, pages 404–411, 1998.
- Sze02. S.M. Sze. *Semiconductor Devices: Physics and Technology*. John Wiley & Sons, 2nd edition, 2002.
- SZK⁺01. A. Stoica, R. Zebulum, D. Keymeulen, R. Tawel, T. Daud, and A. Thakoor. Reconfigurable VLSI architectures for evolvable hardware: from experimental field programmable transistor arrays to evolution-oriented. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(1):227–232, 2001.
- T⁺96. Y. Takahashi et al. Size dependence on the characteristics of Si single electron transistors on SIMOX substrates. *IEEE Transactions on Electron Devices*, 43(8):1213, 1996.
- TBC⁺97. Yuan Taur, D.A. Buchanan, Wei Chen, D.J. Frank, K.E. Ismail, Shih-Hsien Lo, G.A. Sai-Halasz, R.G. Viswanathan, H.-J.C. Wann, S.J. Wind, and Hon-Sum Wong. CMOS scaling into the nanometer regime. *Proceedings of the IEEE*, 85(4):486–504, 1997.
- THML04. M.B. Tahoori, Jing Huang, M. Momenzadeh, and F. Lombardi. Testing of quantum cellular automata. *IEEE Transactions on Nanotechnology*, 3(4):432–442, 2004.
- TL94. P.D. Tougaw and C.S. Lent. Logical devices implemented using quantum cellular automata. *Journal of Applied Physics*, 75(3):1818, 1994.
- TL99. G. Toth and C. S. Lent. Quasiadiabatic switching for metal island quantum dot cellular automata. *Journal of Applied Physics*, 85(5):2977–2984, 1999.
- TMHL04. M.B. Tahoori, M. Momenzadeh, Jin Huang, and F. Lombardi. Defects and faults in quantum cellular automata at nano scale. In *Proceedings of the 22nd IEEE VLSI Test Symposium (VTS04)*, pages 291–296, 2004.
- Var05. Various. GNU multiple precision arithmetic library (GMP). Software library, Dec 2005.
- VCR⁺03. M. Violante, M. Ceschia, M. Reorda, P. Bernardi, M. Rebaudengo, D. Bortolato, M. Bellato, P. Zambolin, and A. Candelori. Analyzing SEU effects in SRAM-based FPGAs. In *Proceedings of the 9th IEEE International Online Testing Symposium (IOLTS03)*, 2003.

- vN56. John von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C.E.Shannon and J.McCarthy, editors, *Automata Studies*, pages 43–98. Princeton University Press, Princeton, NJ, 1956.
- VSC⁺04. M. Violante, L. Sterpone, M. Ceschia, D. Bortolato, P. Bernardi, M.S. Reorda, and A. Paccagnella. Simulation-based analysis of SEU effects in SRAM-based FPGAs. *IEEE Transactions on Nuclear Science*, 51(6):3354–3359, 2004.
- Wak90. John F. Wakerly. *Digital Design Principles and Practices*. Prentice Hall, 1990.
- Wal05. Konrad Walus. *QCA Designer Reference Manual, v2.0.0 (Online)*. University of Calgary, May 2005.
- Wan04. W. Wang. RC hardened FPGA configuration SRAM cell design. *IEE Electronics Letters*, 03:525–526, 2004.
- WFS⁺99. H.-S.P. Wong, D.J. Frank, P.M. Solomon, C.H.J. Wann, and J.J. Welser. Nanoscale CMOS. *Proceedings of the IEEE*, 87(4):537–570, 1999.
- WH95. Michael Wirthlin and Brad Hutchings. A dynamic instruction set computer. In *Proceedings of the 1995 IEEE Symp. on FPGAs for Custom Computing Machines (FCCM95)*, 1995.
- WHG92. J. Welser, J.L. Hoyt, and J.F. Gibbons. NMOS and PMOS transistors fabricated in strained silicon / relaxed silicon-germanium structures. In *Proceedings of the 1992 Electron Devices Meeting*, pages 1000–1002, Dec 1992.
- WJD03. K. Walus, G.A. Jullien, and V.S. Dimitrov. Computer arithmetic structures for quantum cellular automata. In *Conference Record of the Thirty-Seventh Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 1435–1439 Vol.2, 2003.
- WK00. R.S. Williams and P.J. Kuekes. Molecular nanoelectronics. In *Proceedings of the 2000 IEEE International Symposium on Circuits and Systems (ISCAS 2000)*, volume 1, pages 5–7 vol.1, 2000.
- WMSJ05. K. Walus, M. Mazur, G. Schulhof, and G.A. Jullien. Simple 4-bit processor based on quantum-dot cellular automata (QCA). In *Proceedings of the 16th IEEE International Conference on Application-Specific Systems, Architecture Processors (ASAP 2005)*, pages 288–293, 2005.
- Won02. H.-S.P. Wong. Field effect transistors-from silicon MOSFETs to carbon nanotube FETs. In *Proceedings of the 23rd International Conference on Microelectronics (MIEL 2002)*, volume 1, pages 103–107 vol.1, 2002.

- WT99. S.-J. Wang and T.-M. Tsai. Test and diagnosis of faulty logic blocks in FPGAs. *IEEE Proceedings on Computers and Digital Techniques*, 146(2):100–106, 1999.
- WVJD03. K. Walus, A. Vetteth, G.A. Jullien, and V.S. Dimitrov. RAM design using quantum-dot cellular automata. In *Proceedings of the 2003 NanoTechnology Conference and Trade Show*, 2003.
- WWC⁺03. H.C.-H. Wang, Y.-P. Wang, S.-J. Chen, C.-H. Ge, S.M. Ting, J.-Y. Kung, R.-L. Hwang, H.-K. Chiu, L.C. Sheu, P.-Y. Tsai, L.-G. Yao, S.-C. Chen, H.-J. Tao, Y.-C. Yeo, W.-C. Lee, and C. Hu. Substrate-strained silicon technology: process integration [CMOS technology]. In *Technical Digest of the IEEE International Electron Devices Meeting (IEDM '03)*, pages 3.4.1–3.4.4, 2003.
- WWJ03. W. Wang, K. Walus, and G.A. Jullien. Quantum-dot cellular automata adders. In *Proceedings of the 2003 IEEE Nano Conference*, August 2003.
- WWKO05. T. Wei, K. Wu, R. Karri, and A. Orailoglu. Fault tolerant quantum cellular array (QCA) design using triple modular redundancy with shifted operands. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC 2005)*, volume 2, pages 1192–1195 Vol. 2, 2005.
- Xil03. Xilinx, inc. Xilinx defense and aerospace. Powerpoint Presentation, Oct 2003.
- Xil04a. Xilinx, inc. *TMRTTool User Guide (v6.2.03i)*, Sep 2004.
- Xil04b. Xilinx, inc. *Two Flows for Partial Reconfiguration: Module Based or Difference Based (v1.2)*, Sep 2004.
- Xil05. Xilinx, inc. *Virtex-II Pro and Virtex-II Pro X Platform FPGAs: Complete Data Sheet (v4.2)*, March 2005.
- YSN⁺00. A. Yagishita, T. Saito, K. Nakajima, S. Inumiya, Y. Akasaka, Y. Ozawa, K. Hieda, Y. Tsunashima, K. Suguro, T. Arikado, and K. Okumura. High performance damascene metal gate MOSFETs for 0.1 μm regime. *IEEE Transactions on Electron Devices*, 47(5):1028–1034, 2000.
- ZS02. M.M. Ziegler and M.R. Stan. Design and analysis of crossbar circuits for molecular nanoelectronics. In *Proceedings of the 2002 2nd IEEE Conference on Nanotechnology (IEEE-NANO 2002)*, pages 323–327, 2002.

Index

The index is conceptual and does not designate every occurrence of a keyword. Page numbers in bold represent concept definition or introduction.

- W_{in} , 280
- W_{out} , 280
- AMD Athlon
 - ECC codes, 198
- ASIC, 347
- ASIP, 348
- ASRAM, 87
- availability, 333
 - point, **56**
- backward error recovery, 60
- BIST, 55, 99, 178, 195
- bit file, 96, 327
- boundary scan, 335
- boundary scan testing, **54**
- Built-In Self Test, *see* BIST
- bus
 - CAM cache, 223
- bus macros, 335
- cache preloading, 183
- carbon nanotubes, 46
- Cascaded Tri-Modular Redundancy,
 - see* CTMR
- CED, **67**, 82, 178
- cell yield, **85**
- chemical self-assembly, 43
- CLB, 90, **320**
- CMC, **330**
- collection efficiency, **76**
- Concurrent Error Detection, *see* CED
- Configurable Logic Block, *see* CLB
- configuration shifting, 100, 370
- conflicting hit, 183, 218, 219
- constant scaling rules, 14, 78
- coverage, **56**
- critical charge, 76, 78
- CTMR, 63
- decoder, address, 137
- defect clustering, 84
- defect models
 - clustered, 124
 - unclustered, 124
- defect tolerance, 4, **52**, 95
- derating of SER, 77
- design rules, 23, 304
- DICE Memory, 88
- DISC, 351
- DNA self-assembly, 49
- driver contention, 330
- Duplication With Comparison, *see* DWC
- DWC, **67**, 178, 181
 - Instruction, 181
- dynamic reconfiguration, 334
- dynamic routing, 185, 336, 343, 357
- ECC, 77, 80, 132, 198
 - options for cache, 222
 - use in FDT processor, 178
- EEPROM, 318
- error, **51**

- von Neumann, 267
- Error Correcting Codes, *see* ECC
- error detection, **58**
- evolutionary algorithms, 92
- EXE unit replication, 182
- fail-closed fault, 206
- failure, **52**
- Failure Unit, FIT, **57**
- false hit, 218, 219
- false miss, 218, 219
- fault, **51**
 - confinement, **59**
 - detection, 97
 - diagnosis, **59**, 95, 97, 100
 - equivalence, **54**
 - injection, 332
 - latency, 101
 - masking, **59**
 - sensitization, **54**
 - stuck-at, 267
- fault tolerance, 4, **52**, 178
- faults
 - fail closed, 205
 - fail open, 205
 - interconnect, 53
 - von Neumann, 127
- FDT computer, **107**
- Field Programmable Gate Array, *see* FPGA
- flash memory, 318
- forward error recovery, 60
- FPTA, **90**
- frames, configuration, 328
- functional, **56**
- functional fault, 95
- functional fault model, 97
- general purpose processor, 347
- Golay code, extended, 134
 - code choices, 222
 - errors corrected, 198
 - hardware implementation, 218
 - testing, 219
 - use for both soft and hard faults, 223
 - use in CSR, 233
 - use in data registers, 215
 - use in tag registers, 215
- granularity, 320, 322
- half pitch, 12
- Hamming code, 198, 222
- Hamming distance, 219
- hard fault, **52**
- Hardware Description Language, *see* HDL
- hardware/software codesign, 334, 348, 353
- HDL, 325, 354
- IDDQ testing, 98
- incremental routing, 344
- interconnect, 324
- interconnect delay, 21
- ITRS Roadmap, 12
- Joint Test Action Group, *see* JTAG
- JTAG, 54, 190, 330
- logical faults, **53**
- lookup table, **320**
- MADP, 108, 114, 199, **223**
- magnetic RAM, 318
- mapping, 325
- Maximum Allowed Defect Probability, *see* MADP
- memory address range checking, 183
- modular reconfiguration, **130**
- molecular crossbar, 41

- Moore's Law, 1, 11
- MOSFET, **12**
- MTBF, 108
- MTTR, 108

- NAND multiplexing, **64**, 128, 194
- nanoscale CMOS, **12**
- NCD2VHD, 332

- OCR, 358
- operating system
 - fault repair, 184

- packing, 327
- parametric fault, **53**, 95
- parity bits, 58
- partial reconfiguration, 328, 334
- physical damage, 330
- placement, 327, 339
- PLD, **313**
- POST, 191
- power supply voltage, 16, 19, 78
- process size, **12**, 78
- Programmable Logic Device, *see* PLD

- QCA, **47**, 127
 - area
 - clock area limited, **271**
 - cell area limited, **271**
 - area estimation, 270
 - cell fill fraction, 272
 - defects, 267
 - design rules, 304
 - inverter chain, 263
 - lack of TGATES, 269
 - OR arrays, 269
 - switching speed, 262
 - wire structures, 269
 - wires
 - long wires, **272**
 - short wires, **272**

- QCA Designer, 121

- R-Modular Redundancy, *see* RMR
- radiation effects, 72
- rapid prototyping, **314**
- readback, 104
- reconfigurable computing, 69, **347**
- reconfiguration, **68**, 181
- reliability, **56**, 177
- RMR, **61**
- ROCR, 367
- routability, **341**
- routing, 324, 327, 340
- RSFQ, 50
- RSTARS, 101
- runtime, 356
- runtime routing, 102

- scrubbing, 82, 185, **333**, 335
- SEE, 330
- SEFI, **330**
- SelectMAP, 330
- sensitization, *see* fault sensitization
- SER, **52**, 76, 108
- SEU, **73**, 76, 316, 329
 - Cache performance, 238
- short channel effects, 15
- simulated annealing, 339
- Single Electron Transistors, 45
- Single Event Upset, *see* SEU
- SIR, **60**, 181
- soft error, 3, **52**, 73
- Soft Error Rate, *see* SER
- software replacement, 185
- SPICE model, 269
- stuck-at fault model, 97

- supermodule, 278
- system recovery, **60**
- Teramac, 69, 81
- Test and Reconfiguration Controller, *see* TREC
- test coverage, **55**, 96
- test vector, 95
- testing
 - manufacturing, 178
 - off-line, 82, 97
 - on-line, 97
 - VLSI, 53
- threshold gate logic, 132
- threshold voltage, 14, 16
- TMR, **61**
- TMR with Shifted Operands, *see* TMRSO
- TMR-R
 - definition, 132
 - yield, 132
- TMRSO, 268
- TMRTTool, 333
- Tomasulo scheduler
 - fault tolerant, 183
- top level architecture, 187
- TREC, 101
- unclustered defect model, 124
- Versatile Place and Route, *see* VPR
- von Neumann error, 267
- VPR, 341, 345
- yield, **2**, **55**, 85, 95
 - expressions, 125
 - FPGA, 85
- yield equation, 84

REPORT DOCUMENTATION PAGE					<i>Form Approved</i> OMB No. 0704-0188	
The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number. PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.						
1. REPORT DATE (DD-MM-YYYY) 31-08-2006		2. REPORT TYPE Doctoral Dissertation			3. DATES COVERED (From — To) Sept 2003 — Aug 2006	
4. TITLE AND SUBTITLE <div style="text-align: center;">Fault and Defect Tolerant Computer Architectures: Reliable Computing With Unreliable Devices</div>				5a. CONTRACT NUMBER		
				5b. GRANT NUMBER		
				5c. PROGRAM ELEMENT NUMBER		
				5d. PROJECT NUMBER		
6. AUTHOR(S) Roelke, George R., Maj, USAF				5e. TASK NUMBER		
				5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institute of Technology Graduate School of Engineering and Management 2950 Hobson Way WPAFB OH 45433-7765					8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/DS/ENG/06-07	
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) Air Force Research Laboratory, Space Vehicles Directorate Attn: Mr Ken Hunt, (505) 846-4959, ken.hunt@kirtland.af.mil 3550 Aberdeen Ave. S.E. Kirtland Air Force Base, NM 87117					10. SPONSOR/MONITOR'S ACRONYM(S)	
					11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution unlimited						
13. SUPPLEMENTARY NOTES						
14. ABSTRACT This research addresses design of a reliable computer from unreliable device technologies. A system architecture is developed for a "fault and defect tolerant" (FDT) computer. Trade-offs between different techniques are studied and yield and hardware cost models are developed. Fault and defect tolerant designs are created for the processor and the cache memory. Simulation results for the content-addressable memory (CAM)-based cache show 90% yield with device failure probabilities of 3×10^{-6} , three orders of magnitude better than non fault tolerant caches of the same size. The entire processor achieves 70% yield with device failure probabilities exceeding 10^{-6} . The required hardware redundancy is approximately 15 times that of a non-fault tolerant design. While larger than current FT designs, this architecture allows the use of devices much more likely to fail than silicon CMOS. As part of model development, an improved model is derived for NAND Multiplexing. The model is the first accurate model for small and medium amounts of redundancy. Previous models are extended to account for dependence between the inputs and produce more accurate results.						
15. SUBJECT TERMS fault tolerance, computer architecture, reliability(electronics), nanotechnology, very large scale integration.						
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 421	19a. NAME OF RESPONSIBLE PERSON Rusty O. Baldwin, Ph.D., AFIT/ENG	
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (include area code) (937) 255-3636, ext 4445 (DSN 785)	